TEST Automation & Controls

SCADAWARE® DOCUMENT 1010-05
SCADA SYSTEM DESIGN MANUAL

(Preliminary PDF version by AMZ Dec 11 2001)

1 SCADAWARE SYSTEM OVERVIEW

1.1 INTRODUCTION

This document provides basic information on the technical design and operational concepts of TEST INC. SCADAWARE systems. Information on the system hardware components, software features, and system configuration options are presented at an introductory level. The intent is to acquaint technical users not familiar with the system with the features and benefits of the system in typical remote data acquisition situations.

It is important to note that any SCADA system is a *system*, and the proper operation of all components is necessary for the system to do its job. Understanding the functions of each subsystem is essential to the proper application and operation of the unit.

Other documents must be used to supplement the information contained herein. The TEST SCADA PROTOCOL COMMAND REFERENCE contains information on all commands used to operate the program. Also, the hardware manuals for the various configurations must be used to get particular information on the PC, interface boards, modems, printers, and other hardware used in the system.

The SCADAWARE software program was written by Total Engineering Services Team, Inc, New Orleans, La, USA, and is a totally copyrighted work with all rights reserved. The program was developed and written by Arthur M. Zatarain and Corey Moragas for TEST Inc. The program released as of December 1994, was written in Borland Pascal Version 7, in both DPMI and real mode versions. SCADAWARE uses components of the Object Professional and Async Professional Library from TurboPower Software, and graphics primitives from Genus Microprogramming.

Note: This version of the SCADAWARE SYSTEM DESIGN CONCEPTS contains information on TEST's products as of Dec 1, 1994. Versions prior to this date may not have all of the features described in this manual. Contact TEST for information on upgrading an existing system.

1.2 TEST'S SCADA HISTORY

TEST has been installing various types of control and telemetry equipment since 1970. Older SCADA systems were designed around existing RTU and PLC equipment which often wasn't suited to oilfield and industrial installations. Extensive modification and

In almost all cases, the units prior to 1988 were made by a major manufacturer and configured by TEST for each particular installation. Often, the system was affected by compromises that were the result of using a completed unit that best fit the application. It was difficult and expensive to get these standard systems to work in an offshore environment. Sophisticated installations required pre-engineered systems using extremely

expensive PLC and mini-computer based software that were tailored to meet the exact needs of the end user. These systems formed the backbone of the large SCADA systems used by the major oil companies. But the smaller companies could not afford to get into the business of developing a system that would meet their unique needs.

The design of TEST's SCADA systems began in 1987 as an effort to provide a reliable, affordable, and flexible system for typical offshore oil production applications. TEST sought to find a way to allow the development time and costs to be shared by a larger number of users. This would decrease the per-unit cost because the end users would install TEST's system rather than one specially developed for each company.

An effort was made to select an existing, standard SCADA system that TEST could adopt as its own system. Although there were many existing systems on the market at the time, this turned out to be a problem rather than a benefit. It was impossible to find a single product or product line that would work in the wide (and I mean WIDE) variety of installations that TEST must handle. The problem was that they were all too different and required unique engineering and programming for each installation. Each also had a different method of communicating with remote devices. Even different products from the same vendor had different communication requirements. Each supplier worked on his own and provided the features that directly affected his product and its applications without much regard for maintaining compatibility with previously installed systems.

The existing systems had all been based on low level electronic units (RTUs) or control oriented Programmable Logic Controllers (PLCs). These systems had "life cycles" of one to three years, after which they were replaced with a newer model. Although these newer models offered cost and performance advantages over their predecessors, they were usually incompatible in both hardware and software. Therefore, considerable manpower was needed to phase out the old system and integrate the new one. It was actually less expensive to install the more costly older equipment because of the engineering time involved to switch to the newer designs. There are many examples of huge SCADA systems that still exist with relic equipment because the cost of upgrading to newer (but incompatible) equipment is far greater than continuing to buy the outdated RTUs.

With many oilfield and industrial installations, TEST has been successful in meeting the SCADA needs of both small operators and large multi-national corporations. The design philosophy, based on the standard PC, has become accepted as the obvious trend in telemetry systems. TEST is fortunate to have taken this path early on, and is anxious to apply its experience and technology to SCADA applications in all industries.

1.3 TEST SCADAWARE DESIGN

TEST's design strategy was based on the idea that it was time for a radical change in RTU and SCADA system design. The decision was made to start with a "clean slate" and to design a system that would meet the following goals:

. Cost Effective SCADA in a wide variety of oilfield and industrial installations

. Low engineering and programming costs per installation

. Limited hardware obsolescence through the use of generic hardware

. Future expansion without re-engineering or re-programming

. Extensive field programming and diagnostic capability

. Ability to use low cost, dial-up radio and phone communications

The design that emerged from the initial study was an RTU and SCADA system based on Personal Computer (PC) technology. Basing the system on the PC provided a level of field processing power that was unheard of in RTU systems. Because the PC had become a generic item, the availability of components was assured well into the next century. And the continued development of the PC into lower cost and higher performance models meant that a system based on the PC would continue to grow along with the industry.

Once the hardware side of the design was settled, the software options became endless. With the processing power of a PC, almost any type of SCADA protocol could be implemented. A review of past and pending projects indicated that none of the existing protocols provided anywhere near the level of performance that could be derived from a PC based system. The existing protocols had come from the world of discrete electronics, relays, and primitive computer systems. Although they provided some of the needed features, they were totally inept at solving the problems TEST faced on actual field installations. The decision was made to implement a new protocol that would take advantage of the PC to provide a simple, flexible, and reliable SCADA communications system.

This may sound like a harsh attitude in that there were many SCADA systems in operation that worked fine with the older protocols. The main difference in these systems and the ones envisioned by TEST is that the older systems were operated and maintained by skilled technicians. The new TEST system was to be used like the typical PC and would be maintained and operated by normal everyday people.

Existing RTU protocols were extremely complex and served the equipment rather than the users. They were designed to allow primitive hardware to work well at the expense of programming time and maintenance nightmares. TEST SCADA PROTOCOL (TSP), implemented with SCADAWARE, put the PC to work and provided many new features designed to reduce the real cost of the systems: engineering, programming, and maintenance. TSP started with a small command set that provided the basic data transfer and command functions. But the design was such that additional commands could be added in the future that would not make any existing systems obsolete.

This "upward compatible" philosophy has dominated TEST's SCADA systems since the beginning. We constantly add features to the software (and to TSP), but maintain compatibility with any existing system at the same time. So, any system can operate the newer program provided that the installed hardware can support the demands of the application. If it cannot, than a minimal amount of hardware can be changed to provide the additional processing power. The end result is a SCADA system that is much like a PC based Word Processor or Spreadsheet. It just keeps getting better all the time.

TEST's SCADAWARE also works in "Open Architecture" systems which are generally based on the Modbus RTU protocol. This allows a combination of complex TSP and simple Modbus transactions to occur in one system, and provides connection to third party hardware and software. Refer Section 12 for more information on TSP and Modbus interfaces.

1.4 NON-PC BASED RTUS

TEST also makes several smaller RTUs which do not have PC type processors. These units, the 2000 series, use a small microcontroller to replace the Intel 8086-80486 processors used in the larger PC based systems. These smaller systems are suitable for limited function locations where the I/O count is small and there are no local processing needs.

TEST's small RTUs have many of the same diagnostic and other real-world features of the larger systems, but are less expensive to purchase and install. The units can work in either a TSP protocol system or a Modbus system in slave mode. Refer to the specific manuals for each unit for more information on these cost effective RTU solutions.

1.5 TEST INC HARDWARE

The following is a brief description of TEST's SCADA products:

UNIT NAME	PC BASED	APPLICATION
T1000	Yes	High Point count "Smart" Systems
T1200	Yes	Med to Low Point Count "Smart" Systems
T2000	No	Med to Low Point Count "Dumb" Systems and remote I/O
T2200	No	Low point Count "Dumb" Systems and remote I/O
T2250	No	Med point Count "Dumb" Systems and remote I/O
T100	No	Voice output alarm system

2 OVERALL SCADA SYSTEM DESIGN

The TEST SCADA / RTU system is based on Personal Computer (PC) technology using off-the-shelf components whenever possible. A wide variety of configurations are possible, but all share the same PC hardware and software background. This section will explain the basic concept of a SCADA System and how some of the important aspects are handled by TEST's equipment and software.

2.1 PURPOSE OF SCADA SYSTEM

The purpose of the TEST SCADA system is to provide a Personal Computer (PC) based data acquisition system that is located at a remote process location and can report alarms and other data over some form of communications channel. This unit, called the Remote Terminal Unit (RTU), receives input signals from switches and process transmitters located on the equipment to be monitored.

The RTU monitors the signals for alarm conditions and reports back to another computer system, referred to as the "HOST" computer. The HOST is usually just another PC system without any physical I/O attached to it. It receives its input from an RTU (or several RTUs) over a communications channel. The update to a HOST from an RTU can be initiated by either the RTU or the HOST. The updates can occur in periods measured from seconds to hours or even days. The frequency depends on the communications system as well as the requirements of the application.

Because the HOST is not physically connected to the actual I/O points, it simply maintains an image of the RTU status. This image represents the state of the RTU at the time of the last update. Therefore, displays and reports generated by the HOST may not exactly correspond to the state of the actual field points if an update has not recently occurred.

Aside from the remote I/O, the operation of a HOST computer is almost the same as that of an actual RTU. The system maintains a list of points, per logical RTU, exactly as the points are physically attached at the RTU. The HOST can be used for control by sending commands to the RTU to turn outputs off or on. In some instances, the outputs can be controlled automatically by the RTU without any connection to the HOST. This control is normally associated with process adjustments, equipment control, or reset functions.

The Host computer may also have some local data points that are used for various sequencing or control functions. For example, the Host may have timer channels that determine how often certain functions should occur. These Host timers are handled exactly like timers at an RTU because the same program operates at all locations. Other Host related channels may include production totals, flow averages, or communications failure indicators. So, TEST's system allows for RTU type functions at the Host as well as Host type functions at the RTU.

2.2 DATA FLOW

The data flow from the field transmitter or switch to the printed report at the Host happens in several discrete steps. Each step is independent of the others, and must happen in the proper sequence in order for timely data to be delivered to the Host system. The data can be examined at each step to verify and diagnose system operation.

back of old brochure

The data path from the field is as follows:

1 Field Transmitter: This is where the data originates within the process being monitored. Verification of these signals requires measurement in the field and at the connection point to the RTU. The signals are generally switches, voltages, (1- 5VDC) or currents (4-20ma) representing actual process values.

2. I/O Interface: The field signals are converted from voltages into computer data by the I/O interface system. TEST SCADA systems are capable of using different types of I/O devices. These devices simply provide the raw data conversion and are not directly involved in alarm sensing, which is done within the PC.

3. RTU PC Memory: The converted signals from the I/O interface system are read by the processor, converted to engineering units, and stored as numbers in the processor's random access memory (RAM). The points are scanned many times per second, and the processor stores both current values as well as historical values. These are the values displayed on the screen at the RTU (assuming a CRT is connected).

4. Communications Line: The data is sent from one system to another in digital format over the communications line. This is the most difficult section of the data flow to control because it often relies on a less than reliable link. The data must pass through several phases:

   1. Digital to tone conversion by the sending modem.
   2. Injection into communications line by sending modem.
   3. Transmission of the signal by the communications system.
   4. Reception of the tone signal by the receiving modem.
   5. Conversion of the tone back into digital data.

   This process occurs instantly in most systems, although some radio and satellite systems have slight delays that can cause timing problems.

5. HOST PC Memory: After reception over the communications line, the data appears in the Host memory exactly as it appeared in the RTU memory. While the RTU memory contains continuously updated data, the host data represents the state of things at the time the data was last received.

6. HOST Display or Printed Report: A display or report on the Host is made from the last known data, and the time of the update is shown on the display or report.

2.3 RTU POLLING

Polling is the process where one unit (Host or RTU) contacts another unit and exchanges data. The types of polling supported by the system include:

Poll after a number of seconds (using Timers).
Poll at specific times of the day (using Agenda).
Poll on demand by operator (using F5).
Poll on change of alarm status (channel setup).
Poll in continuous loop (command file).

The RTU will initiate a poll to the HOST whenever the RTU goes into an alarm state from a channel that is setup to cause callouts. After completing the POLL, the RTU should process an Acknowledgement command and thereby take itself out of alarm mode.

2.4 TEST SCADA ADVANTAGES

A major difference in the "smart" TEST SCADA systems and other systems is the level of processing power located right at the remote location. Traditional SCADA systems require the HOST computer to continuously call, or "poll", the RTU in order to detect the alarms. In those cases, the RTU merely acts as a means to connect field devices to the HOST computer. The TEST system, however, allows the RTU to have the same processing capabilities as most HOST computers. This has several advantages:

1. The RTU occupies costly communications channels only when necessary, eliminating constant polling requirements.

2. Calculations such as AGA-3 Gas flow can be done directly at the RTU. Alarm points can be specified in actual flow units rather than values related to the variables of the equation. This means that a low gas flow alarm can be specified in MMCF rather than in inches of water across the orifice plate.

3. The RTU or HOST can store information permanently on various types of disk storage devices or a printer. This Data logging and storage can be provided to allow continuous recording during periods of communications failure common in remote locations. Alarm conditions, data values, or text messages can be stored as well as transferred to other computers for later processing.

4. Calibration and maintenance is simplified by providing the power of a full PC right in the field. This is done by providing diagnostic and calibration procedures on the RTU and eliminating the need for the HOST computer to evaluate data values.

5. Some control functions can be provided locally by the RTU without any contact with the HOST. Status outputs can be controlled in response to many types of events such as alarm conditions, timer timeouts, or communications failures.

2.5 TYPICAL APPLICATIONS

The basic use of TEST's SCADA systems is to monitor a remote location, such as an unmanned production facility, and have the RTU report back to a HOST computer. The HOST can be at a nearby manned facility or at a distant location. Installations such as this normally use low power radios to communicate between facilities. These radios can be the normal voice radios used in the field, or can be on separate frequencies to reduce noise on the normal channels. Of course, the system can also use phones or direct connections to transfer information among the units.

The purpose of such a system is to provide alarm reports as soon as possible so that personnel may be dispatched to correct any problems. Without SCADA, the facility may require frequent visits for routine checks.

Another use of the SCADA system is to allow control of the remote facility from the main facility. This control can be for shut-in operations, well control, or other process sequencing as allowed by MMS[1] regulations.

This system may be further enhanced by the addition of a phone link from the main facility to a distant location. This allows for the main and remote facilities to be monitored and controlled from the distant location at times when the main facility is unattended. This is often the case during hurricane evacuations or periods when production is curtailed due to market conditions.

The HOST system can also act as an RTU by adding I/O interfacing at the host location. Because there is not much difference between an RTU and a HOST in the TEST systems, RTU type operation at any location is possible. The main differences are in the programming and setup options that determine how the system reacts to abnormal conditions.

[1]MMS - Minerals Management Service, US Department of the Interior

3 RTU AND HOST HARDWARE COMPONENTS

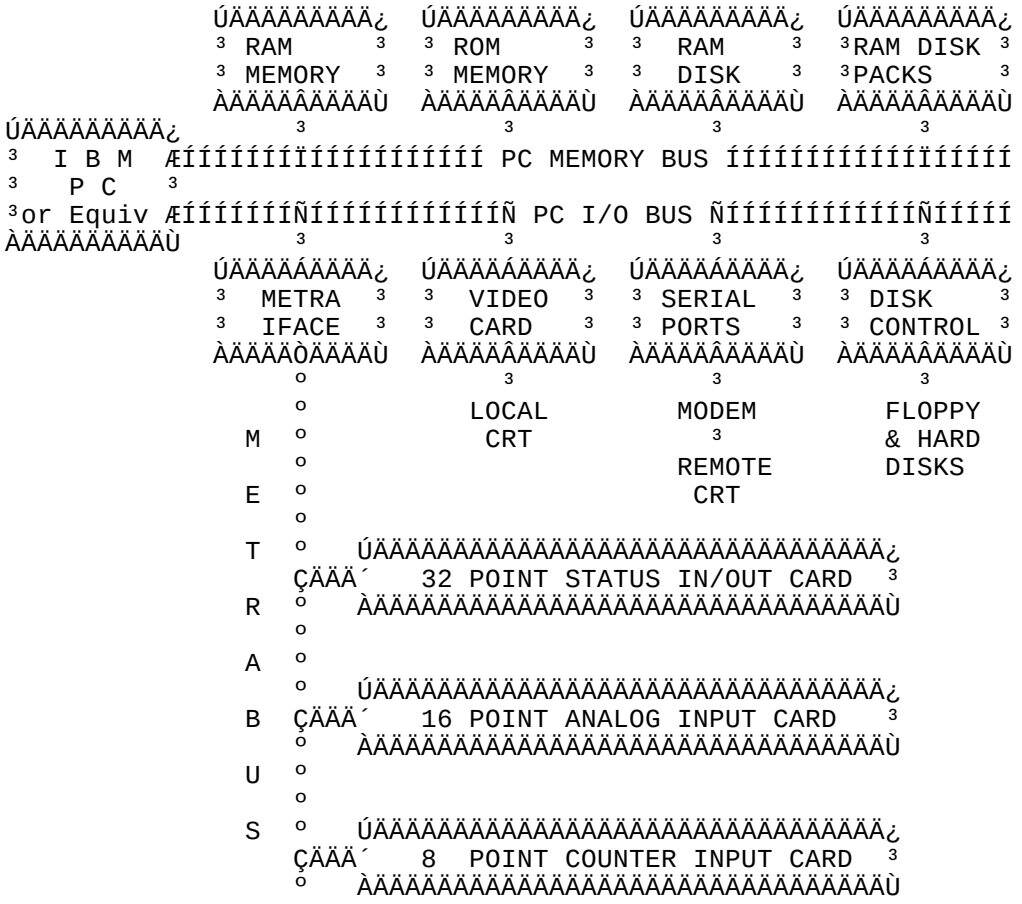
3.1 PERSONAL COMPUTER

The TEST SCADA system is based on personal computer technology using many common PC hardware components and techniques. A summary of the hardware components is as follows:

1. PC: Contains the computer portion of the system including CPU processor, ROM and RAM memory, disk storage, video display, keyboard, and process I/O interface. The RTU/SCADA software operates within this component.

2. PROCESS I/O: The input and output devices that connect the physical world to the computer. This includes the computer interface as well as the field transmitters and switches. The I/O devices currently supported are "Metrabus" compatible interface cards, relay type input and output cards, and many generic PC based I/O cards.

3. DATA COMMUNICATIONS: A comm port (serial or network) is used to convert digital computer information to signals that can be transmitted over a wire, voice grade phone line or radio channel, or a computer network.



TYPICAL METRABUS BASED RTU CONFIGURATION

4. POWER SUPPLY: Receives either 120 VAC or 10-40 VDC and converts it into the necessary PC operating voltages (+5, +-12 VDC). An additional 24 VDC supply can also be provided to power 4-20ma current loop type transducers. Remote locations may use Solar cells to recharge the batteries which power the system.

The HOST and RTU computers are very similar in design and function because they are all basically Personal Computers. Although they are all very similar, each always has one or two features that make it unique to that particular installation. The main difference between an RTU and a HOST computer is that the HOST normally has no actual process I/O devices connected to it. It receives values from the RTU over the communications channel. Once the values are received, they are processed exactly as they were in the RTU. In all other respects, the HOST configuration and operation is identical to the RTU. A typical configuration for an RTU using Metrabus compatible interface cards is shown above.

3.2 PC COMPONENT

IBM PC/XT/AT computer. The selection of the actual processor is based on processing requirements, electrical power requirements, and the needs of other computer applications running on the same system. Most RTU installations can be accommodated with a 8Mhz PC/XT type system or higher. With the exception of the disk devices and video, the operation of the system is identical on all hardware configurations.

The PC based RTU provides the capability to scan the process I/O, generate displays, make alarm decisions, and report data to another computer. The traditional PC features like video displays, keyboards, serial and parallel ports, DOS operating system, disk storage, and expansion bus cards are all available for use by the SCADA program as well as other PC based software. *Note: The use of a PC for an RTU prevents concurrent use by other programs. The use of a PC as a Host is possible although the SCADA program cannot be run at the same time as other applications. Therefore, an RTU needs a dedicated PC while a Host PC can be used with TEST's program as well as other programs.*

3.3 PROCESS I/O HARDWARE

All SCADA systems require some means of obtaining switch status and analog value data for processing by the computer component. The TEST SCADA system supports I/O devices such as the "Metrabus" family of interface boards manufactured by The Metrabyte Corporation, simple relay type input and output cards that are installed directly into the PC I/O bus, and other generic PC based I/O cards. The details of these systems are provided in separate documents prepared for each system type.

The typical I/O system includes the following I/O types:

1. Status (digital) inputs and outputs: These are discrete off-on inputs and outputs. The I/O receives inputs from devices like switches or electronic levels, and controls outputs like control relays, lights, and solenoids.

2. Analog Input: These inputs are typically voltage levels that can vary from some minimum to a maximum level. The minimum is called the "Zero" point, and the maximum is called the "Span". The I/O hardware and a software driver convert the voltages into 16 bit signed numbers that are further manipulated by the SCADA software. If the analog input hardware is less than 16 bits (normally 12 bits), then the software driver will scale the value up to 16 bits. In this manner, all analog inputs look the same to the SCADA software regardless of their actual resolution.

3. Counter Inputs: Counter inputs are used to interface transmitters that generate pulses that relate to rate or total flow. The pulses can be slow (one pulse every second or so) indicating totalized flow, or can be fast (hardware dependent, but often thousands per second) indicating rate. Totalized pulses often originate from instruments such as gas flow computers, while rate pulses originate from turbine meter magnetic pickups.

3.4 DATA COMMUNICATIONS

SCADA systems normally require the use of some type of communications medium to send data from the RTU to the Host computer. This has traditionally been some sort of "on-line" communications link using a dedicated phone line or microwave channel. The TEST SCADA system, however, can use either telephone or radio communications on a "dial up" basis to minimize connect time and its associated costs.

Using a standard serial communications port on the PC (COM1- COM4), a modem of the proper type connects the computer to the communications line. The modem changes the digital computer data into audio tones that can be transmitted much as any other sound is carried on the line. A similar modem on the receiving end reconverts the tones back into digital data for use by the other computer.

Various types of intelligent and dumb modems are used in the SCADA system depending on the type of communications line being used:

1. HAYES Type Phone Modem: The Hayes standard for phone modems has produced a variety of low cost, high performance modems that are essentially identical in operation. The TEST SCADA system can use almost any compatible modem, although some have specific advantages (like 12 VDC operation) that promote their use in certain applications.

2. Packet Radio Controller: The PRC is a special modem device that allows connection of the computer to a voice type radio. The unit takes care of message routing and error checking so that many units can share a common radio frequency.

3. Simple and Multi-Drop Modems: Simple modems which only convert digital data into tones can be used in point-to-point applications. Multi-Drop modems, which turn the transmit carrier off and on, can be used when many stations share a single communications path.

4. Direct Connections: A modem is not actually required as two (or more) units can be connected by direct wire if the proper serial ports are available. RS-232 ports allow only two devices, while other RS-xxx ports allow up to 50 units to share a single wire.

Many types of modems are actually micro-processor based devices that are themselves small computer systems. Their processing capabilities are specifically designed for communication tasks, and they take care of many dialing, timing, call sign, and other low level tasks. Consult the operations manual for the specific modem being used for information on its command processing capabilities.

The SCADA program provides special commands (HAYES and PRC) to assist in programming some types of modems. These commands take care of the sticky details of getting the modems in command mode and then returning them to an on-line state if needed.

For example, a Hayes modem has the command codes Q1 and E0 to set the modem in Quiet mode and to turn character Echo off. To pass these parameters to the modem, the command line HAYES Q1 E0 can be used. Similarly, the Packet Radio Controller has a command to set its radio address called MYCALL (short for my call sign). To set a PRC's address the command PRC MYCALL anyname could be used. These commands are most often used in the STARTx and BYEx command files, although they can be fed to an RTU task in any other fashion as well.

Telephone systems used by the SCADA system can be almost any type that provide a standard Bell RJ11 (modular phone) jack. The communications rate is limited by the quality and reliability of the channel, although the software is designed to handle poor communications. Generally speaking, the telephone line is the weakest part of most remote SCADA system installations.

Radio systems can be any type that can provide an adequate bandwidth for the modem. FCC limitations may restrict the amount of data transmitted at any one time. Radio systems, once installed properly, generally

provide less trouble than phone systems due to the consistency of the equipment and signal levels.

3.5 POWER SUPPLY

The PC, I/O subsystem, communications devices, and other system components require a number of regulated voltages in order to operate properly. The Power Supply is a standard component in a desktop PC, while industrial PC systems may require a special power supply to accommodate battery or Solar power operation. For example, the TEST SCADA system uses a power supply that accepts either 120 VAC or 10-40 VDC (but not both) and produces the +5, +-12 needed by the RTU hardware system.

In the case of battery powered systems, solar panels are often used to charge the storage batteries. If possible, one of the status or analog input channels is used to monitor the voltage of the batteries to avoid any downtime due to low battery power.

The power requirements of the SCADA system fall into 2 general categories:

1. Processor and I/O: Continuous low power requirement.

2. Communications: Periodic high power and continuous low power requirement.

Battery powered systems must be designed to accommodate the periodic requirements based on the anticipated communications usage.

[power supply slide]

5 COMPUTER SETUP, STARTUP, AND OPERATION

5.1 PROGRAM AND DATA FILES

In addition to the main RTU/SCADA program file, SCADAWARE uses a number of data files to store setup, channel program information, and automation tasks. These files are normally contained in the same DOS directory as the program file, although this is not required.

The complete version of SCADAWARE is contained in a program titled RTUMON3. The limited version is called RTULITE3. The programs are very similar in function, but internally are very different. RTUMON3 works as a DOS Protected Mode Interface (DPMI) program, while RTULITE3 is a simple real mode program.

The full version has all program features, while the "lite" version is missing advanced features in order to reduce program size and complexity. Startup of both types is similar, although there are several differences explained here. In almost all cases, the program name RTULITE3 can be substituted for RTUMON3 without any other changes.

5.2 DOS CONFIG.SYS FILE

Warning! Some knowledge of PC boot sequences is necessary to fully understand this section. It is presented as a guide for experienced PC users when setting up the RTU or Host PC Computer.

All computers operating the RTU/SCADA program should have a config.sys file present on the boot device. This file will tailor the fundamental properties of the system to meet the needs of the RTU program. In spite of SCADAWARE's real-time, multi-tasking capability, it operates as a very tame DOS application which conforms to rigid DPMI specifications for memory use and processor allocation. In order for SCADAWARE to operate properly, the PC's Config.Sys file should have at least the following lines:

```
CONFIG.SYS
DEVICE=HIMEM.SYS  Microsoft DOS Extended Memory Manager
FILES = 24        Minimum Open files
BUFFERS = 20            Typical file buffer count
```

The older RTUMON2 program consisted of two files, RTUMON2.EXE and RTUMON2.OVR. The full SCADAWARE program does not use overlays. SCADAWARE Lite still uses overlays, but stores them in the main program file itself rather than in a separate overlay file.

The full RTUMON3 program requires several additional support files that the RTUMON2 program did not. They must be located in the RTU program directory:

DPMI16BI.OVL	Borland's DPMI Server (interfaces with HIMEM.SYS)
RTM.EXE	Borland Pascal's run-time manager.
RTUFONT.LIB	Genus Graphics Package Font and Shape library

(Note: these extra files are not required for the Lite version.)

The entries in the file that are needed for the RTU program should be added to those necessary for other features or programs on the computer. These entries will vary according to the nature of the hardware implementing the system. For example, ROM and floppy based systems normally require that a RAM disk be installed via the config.sys file to speed up RTU command file processing.

Systems being used in different countries may need to include a line in the config.sys file to control the date format. The following commands can be used in the config.sys file to cause the dates to appear as follows:

COUNTRY = 001 <-- United States mm/dd/yy (DOS default)
COUNTRY = 003 <-- Latin America dd/mm/yy

A typical config.sys for a ROM based system is as follows:

```
shell = a:\command.com /f /p <- reconfigure COMMAND.COM
buffers = 10     <- file transfer buffers
files = 16      <- max open files
device = ansi.sys    <- Terminal driver
device = ssd-drvr.sys X1E  <- Drive C:
device = ssd-drvr.sys X1A  <- Drive D:
```

5.3 SCADAWARE VERSIONS

RTUMON3 and RTULITE3 are revised versions of TEST's earlier program releases. Some changes are necessary to implement the Release 3 programs on systems running older software. Although these changes are minimal, they are necessary for proper operation of the TEST SCADA program. The main difference between the old and new programs is that the internal structure has been modified to work under the command of a revised Multi-tasking program manager. The full version of the program, RTUMON3, also uses the DOS Protected Mode Interface (DPMI) to access the advanced features of the Intel 80386, 80486, and later processors which are required for its operation.

SCADAWARE Lite, RTULITE3, does not use protected mode and will therefore run on an 8086 or 80286 processor. RTULITE3 will use XMS memory to a limited degree and is therefore useful on some larger processors that do not need the advantages of the full SCADAWARE program. The advantages and limitations of Lite version are listed below:

RTUMON3.EXE - PROTECTED MODE VERSION

1. Requires 80386, 486, or later CPU.
2. Requires at least 2Mb of RAM with at least 1Mb of free XMS memory.
3. Math Coprocessor is not required, although it will be used if present.
4. Has Database and Graphic capabilities.
5. Has Modbus, ROC, and other Open Architecture features.
6. Can handle up to 250 RTUs and 24 tasks per computer.
7. Real Numbers are stored in IEEE ANSI standard 4-byte format
8. Integers are stored in IEEE ANSI standard 2 and 4-byte format.

RTULITE3.EXE - REAL MODE VERSION

1. Small Disk Space Requirement (less than 500K total for RTULITE3.EXE).
2. No need for 80386 or later CPU.
3. Main Memory Requirement is 640K with at least 560K free.
4. Extended Memory (XMS) is not required, although it will be used to store task stacks.
5. Math Coprocessor is not required or used.
6. No Database or Graphic capabilities.
7. No Modbus, ROC, or other Open Architecture features.
8. Simplified Image Save System with no Network Support.
9. Limit of 10 RTUs and 8 tasks per computer.
10. Real Numbers are stored in Borland 6-byte format.

In most respects, conversion from older programs is the same for SCADAWARE or SCADAWARE LITE. Only SCADAWARE will be addressed directly, with any special requirements for SCADAWARE LITE mentioned where required.

5.4 AUTOMATED COMPUTER STARTUP

Most installations are setup so that the computer automatically runs the RTUMON3 program with the proper parameters each time the system is booted. The standard way of doing this is to have the computer's normal autoexec.bat file execute a command called START as its last line. This command is actually a batch (.BAT) file in the program's root or \RTU directory.

The START.BAT file will contain any DOS commands necessary to start the program, including the RTUMON3 command that actually starts the SCADA program. If desired, the START command can be entered in the batch file after the RTUMON3 command so that the program always restarts if somehow terminated. This is fine for unattended locations, but may be a nuisance in other locations because an endless loop is established that continuously runs the RTUMON3 program.

An example autoexec.bat (with comments) for a HOST installation with a hard disk may look like:

```
prompt $P$G      ; tell user where he is
path C:\; C:\DOS; C:\util  ; Tell DOS where to look
cd \rtu       ; switch to program's dir
START       ; start the RTU program
```

This file, which is always processed by DOS when the PC system is restarted, does some housekeeping and then starts the program directly. When SCADAWARE terminates, it will be automatically restarted because the file ends with a START command.

An example autoexec.bat (with comments) for an RTU installation with a ram disk C: may look like:

```
setwait UMB32 LMB512 UMW1  ; Special setup for particular PC
set comspec=A:\command.com  ; Tell DOS what to use
path a:\      ; Default search path
pcclock       ; Read watch and set time
:looper       ; Program loop start
a:
cd \
locate start.bat c:\ d:\ e:\ ; Try and find start file
if errorlevel 99 goto looper ; If not found at all, loop
if errorlevel 3 goto use_e  ; if found on drive E
if errorlevel 2 goto use_d  ; if found on drive D
if errorlevel 1 goto use_c  ; if found on drive C
goto looper      ; keep searching
:use_c

c:        ; Switch to C:
cd \       ; Root on C
start       ; Execute START.BAT to run RTUMON3
goto looper      ; Loop again
(additional code for each drive follows here)
```

Notice that this autoexec file does the normal housekeeping and then executes a special TEST prepared program called "LOCATE" to find the start.bat file. The program looks in the drive sequence C D E specified on the command line. The advantage of this special program is that it avoids "disk not ready" type errors that will usually cause similar programs to crash.

The point is to find the first working disk device that contains the program we want. The location is returned in a special DOS variable called errorlevel, and the program branches according to its value. The value of 99 indicates no file was ever found, so the autoexec just loops till satisfied.

This autoexec.bat file is typical of the ROM based RTU systems where the location of the startup file is not known. It may be on a RAM disk, a 5-1/4" diskette, a 3-1/2" diskette, or a ROM disk. This type of autoexec.bat file allows for flexible operation without changing the ROM disk by letting the autoexec find the first acceptable disk from which to run.

Once the start.bat file is located, control is turned over to it by the autoexec file. An example START.BAT file for a floppy based RTU that uses a volatile RAM disk F: is as follows:

```
copy *.rtu F:     ; copy command files to RAM disk
RTUMON3 /f=wc458a    ; start program for wc 458a
start       ; loop forever
```

This example start.bat copies the command files (.RTU) needed for normal operation to the ram disk called F: (which must have been established in the config.sys at boot time). The use of the ram disk allows for very fast command file execution without the power consumption and delays associated with the floppy disk drive. RTU systems running from a permanent RAM disk do not normally need this option.

5.5 ROM BASED DISK DRIVES AND RTU BOOT SEQUENCE

The RTU computer can be any type of PC compatible system that will boot according to normal PC standards. However, some installations use special disk devices located on Read Only Memory (ROM) chips or possibly on Non Volatile Random Access Memory (NOVRAM) boards. These devices emulate normal magnetic disk devices but are completely solid state and have no moving parts.

In systems with a boot ROM, special instructions are placed in the autoexec.bat file to allow a variety of boot sequences to occur without changing ROMs. A key to this process is a TEST prepared program called LOCATE which allows the start sequence to find the desired programs without running into problems with non-existent or not ready disk devices. This program looks for the initial RTU program called "START.BAT". This is a normal DOS batch file that must exist somewhere in the search path in order for the system to load. If not found, the autoexec.bat file will keep looping till one is found. It can be aborted with the usual control-C to allow manual operation.

The ROM will set up disk devices in the RTU according to the following schedule:

DISK	ACTUAL DEVICE
A:	Boot ROM itself.
B:	First Floppy drive
C:	Novram device
D:	ROM Drive on Memory Expansion Card

During the processing of autoexec.bat, a search is made for the file START.BAT in the following directories.

A:\ <- ROM Disk (never finds it there!)
C:\ <- NOVRAM
B:\ <- First Floppy
D:\ <- Second ROM Drive

The first disk drive that is operable and contains the file START.BAT will become the default system drive (i.e. the one that shows up at the DOS prompt) and execution will continue with START.BAT on that directory.

This allows for a flexible startup because the operator can remove and insert a variety of devices and have the system boot from any device containing the program "START.BAT". If more than one contains the program, the first one found in the above search sequence will get control. If you want to skip over one of the drives, remove the NOVRAM or diskette and the startup process will skip over it.

startup slide

5.6 RTUMON3 PROGRAM STARTUP

No matter how the RTUMON3.EXE program is loaded, the sequence of events that occur after the load will be the same. The RTU/SCADA program will maintain control of the system until the program is terminated. At that point, the START.BAT file that had loaded RTUMON3 will continue. If the program was simply started from the DOS prompt, then control is again returned to the prompt.

Just after starting, RTUMON3 signs on with the program version number. The program then looks for the main configuration file (default is RTU.DAT), containing special task setups. If found, this file will tell the multi-tasker how many tasks to setup and what to call them. If not found, the program will halt. The default file name can be overridden by the /F= command line option discussed below.

The program then initializes the tasker and clears a number of internal sub-systems. The communications channels are then initialized to default values
(which may be overridden during later processing).

In the main configuration file just loaded, the DRIVER task setup is used to specify which types of I/O devices are being used. Each type has a separate configuration file associated with it, and these files are processed at this time. If a file name is not specified in the TASK DRIVER line, the file name "RTU" is used with an extension appropriate for each I/O device. For more information about the TASK DRIVER command refer to section 7.11 of this manual.

After all this setting up, the multi-tasker is started so that task switching can occur. HOWEVER, ONLY THE LOCAL TASK NUMBER 0 IS AUTOMATICALLY STARTED when the RTUMON3 program starts. The starting of task 0 will cause it to look, by default, for the file START0.RTU, which must contain all of the other steps necessary to start all other scheduled tasks. A different task 0 startup file can be specified with the /S= command line option discussed below.

5.7 RTUMON3 COMMAND LINE OPTIONS

It is possible, and desirable, to override the default file names used to initially start the system. This allows for files with names related to the particular installation to be used rather than generic names like "RTU.DAT". For example, the main file for a facility called WC458 can be named WC458.DAT. This also simplifies testing of Host and RTU locations because numerous files can exist in the same directory during setup.

The primary means of specifying options are with command line switches. This technique is used on many DOS programs and the method is very similar in all cases. A switch character (the slash "/" character, not the backslash "\" character) is used to signal that an option is being provided on the command line. Information related to the option follows the switch character, and one switch character is required for each option being specified on a single line.

In the most basic case, entering "RTUMON3" on the command line is all that is needed to start the system. If options are provided, they must follow the RTUMON3 command after at least one space. In our case, the options consist of a letter and an equal sign followed by a file name. The letter signals the purpose of the option, and the file name is the name that will override the specific default file name assumed by the program.

The available options are:

/F=filename - Main config file to replace RTU.DAT. *RTUMON3 /F=MYRTU.DAT*
/S=filename - Replacement for START0.RTU. *RTUMON3 /S=SPECIAL.RTU*

A parameter is available to allow selection of the system tick interrupt used by SCADAWARE. This topic is covered in section 4.9 of this document. The new parameter is /T and has the following options:

/T=H Hardware Timer Tick, PC interrupt 8 at 18.2 per second.

/T=D Dos Multiplex Interrupt 1C hex at 18.2 per second
/T=S Soft interrupt called as often as possible with average rate at 18.2 per second.

Another parameter is available for setting the address of the security lock (dongle). Details on this option (/D=) are provided below.

/D=1 LPT1 (address determined by bios)
/D=2 LPT2 (address determined by bios)
/D=3 LPT3 (address determined by bios)
/D=5 Physical Port address 03BCh
/D=6 Physical Port address 0378h
/D=7 Physical Port address 0278h

The following applies only to RTUMON (full program version):

/PLAY Install Sound Playback system

The following apply only to RTULITE:

/O=filename	- Replacement for program overlay file	*RTULITE3 /O=D:RTU.OVR*
/B=bufsize	- Adjustment to default overlay buffer size	*RTULITE3 /B=40000*
/E=xx	- Reserve xxK Memory for Exec Shell.	*RTULITE3 /E=45*
/X	- Do not use XMS even if available	*RTULINE3 /X*

Note that more than one command line option can be (and often is) used on a single DOS command line. The line is normally part of a standard start sequence that is stored in a DOS Batch file called START.BAT. So, to start the program and have it process VR167A.DAT rather than RTU.DAT, and to start task 0 with SPECIAL.RTU rather than START0.RTU, we would enter:

RTULITE3 /F=vr167a.dat /S=special.rtu /B=32000

Note that upper and lower case does not matter in this case. The program also assumes a file type of DAT for the /F option.

Normally, the proper command line is placed into a DOS batch file (called START.BAT) along with any other special startup tricks needed for the program. These tricks may include defeating any existing TSR programs, copying files to a ram disk (on floppy based systems), and changing the DOS prompt. A sample start.bat is as follows:

```
copy *.rtu C:        <- ram disk
copy *.dat C:        <- ram disk
c:              <- change drives
prompt Hit EXIT to return to RTUMON3$_$P$G <- DOS prompt
b:RTULITE3   /F=c:wc146.dat /o=b:RTULITE3.exe <- program start
prompt $P$G         <- restore prompt
start           <- do it all over
```

5.8 LOCATING SCADAWARE LITE PROGRAM FILES

The main component of the Lite program, RTULITE3.EXE, can be located on any legal DOS disk device. This allows flexibility in using RAM, ROM, and floppy disk drives to optimize program performance. When using SCADAWARE Lite, it is necessary that both DOS and the program itself be informed of the proper location of the main file because it is accessed during normal program operation to retrieve overlay modules. Identification of the program location takes place in the START.BAT file used to kick off program execution.

For example, suppose we have a typical RTU with a boot ROM as drive A:, a floppy disk as drive B:, a RAM drive as C:, and a ROM drive as D:. We want to operate the system using the RAM drive as the default device, but have the program and overlay located on the ROM drive D:. During the startup sequence, the boot ROM A: will search for a file called START.BAT, and when found, will switch to that drive before executing that file.

In our example, we want to use drive C: as the default drive where all the RTU and other setup files will be located. So, we will put START.BAT on drive C:. This file will contain the following typical information:

```
prompt ENTER EXIT TO RETURN TO RTU PROGRAM $_$P$G
c:
d:RTULITE3 /o=d:RTULITE3.exe /f=rtuname.dat
prompt $P$G
start
```

Each line has a specific purpose when executed by DOS. The first line changes the DOS command prompt to provide a message telling the user how to get back to the RTU program. This message will only appear when the user "shells out" of the RTU program. The second line switches operation to use C: as the default disk drive.

The third line is the one that starts the actual RTU program. The first entry on the line tells DOS to load the program RTULITE3.EXE from drive D: rather than from the default drive, which is currently C:. This is all DOS needs to get the program started. The remaining entries on the line are for the RTULITE3 program's internal use only. The /O= parameter tells RTUMON3 where to find it's overlay routines, on drive D:. The /F= parameter would have the name of the control file to be processed on startup.

The fourth line will be executed by DOS when the program terminates. It restores the prompt to the normal **C:\>.**

The fifth line restarts this START.BAT program all over again. This is normally done on RTUs to keep the program in a never ending loop.

Systems using a hard disk or other large primary device need not relocate anything. Simply put everything in one directory and start the program as follows:

RTULITE3 /f=rtuname.dat

Once the program starts up, it assumes that all the necessary RTU, MNU, and any other files will be located on the default disk unit (C: in these examples). No disk letter is needed before the RTULITE3 program name because the default of C: will be assumed by DOS. As you can see, there is a lot of assuming going on, and it is important to keep track of what is where, and what the various components expect to find.

6 MULTI-TASKING

6.1 MULTI-TASKER SUPERVISOR

The RTU/SCADA program contains a proprietary Multi-Tasking Supervisor (MTS) which is used to control the simultaneous operation of several functions on a single computer. When using the RTU/SCADA program, it appears that several independent functions, called tasks, are operating simultaneously. In reality, only one task is operating at any given instant (measured in milliseconds). This program uses different tasks to perform specific functions which contribute to the overall scheme of operation.

Because there are many tasks and only one processor, all tasks must share the processor on a time sharing basis. Each task gets a certain amount of processor time to do whatever it needs to do, then it must stop processing and pass control to another task. Eventually, each task will regain control of the processor and continue processing exactly where it had left off. When its time is up, a task will once again stop processing, pass control to another task, and begin waiting for its turn to continue processing. The computer processes commands so quickly that the time a task spends waiting to regain processor control goes virtually unnoticed.

The "round-robin" style of task swapping is a gross simplification of what actually occurs inside the program. Some task functions are given special priorities, most notably the *SYSTEM* task, which takes care of time critical functions within the program. Another special task, the *SCAN* task, gets the opportunity to run *on each and every clock tick* to take care of I/O related functions that occur very quickly (i.e. counter inputs).

Message management is also an important part of a multi-tasking supervisor. In TEST's system, a message is a text line containing a TSP command of any type. Tasks "read" these messages from various sources including the keyboard or serial port, files, libraries, and inter-task messages. The MTS provides for the proper transfer of messages from one task to another, and also provides the mechanisms to store messages in a list (queue) until a task is ready to process them. It is very important that the proper task get each message because a task meant for a serial port task will not necessarily make sense to one that manages the local keyboard.

All of these functions occur automatically and are mostly hidden from the casual user. However, those interested in setting up or programming complex SCADA applications must understand the MTS and how it manages the complete SCADA system. *Multi-tasking is not a trivial subject.* However, the concept is critical in understanding the power of TEST's SCADA system and TSP protocol.

Think of a *TASK* as a single high level function within the program. For example, the portion of the program that interacts with the local user is a single task (called the Local task, or Task 0). This unit receives your keypresses, presents displays, edits files, and performs other human oriented duties. Another separate task (or series of tasks) takes care of scanning the input/output devices to read status and analog values. These DRIVER tasks have nothing to do with the Local task just described. Their functions are completely independent (well, almost). When a task is executing, it "thinks" it's the only program on the computer. When they are suspended by the multi-tasker, they effectively go to sleep and resume operation just where they left off when it is again their turn to run. The job of the multi-tasker is to coordinate the actions of these various tasks so that each gets adequate processor time without "hogging" the systems's resources.

6.2 A MULTI-TASKER ANALOGY

Its analogy Time! This section will offer a simple analogy for a Multi-Tasker that will help non-programmers grasp the concept. If you are a computer guru, feel free to skip this section. But, if the idea of a CPU juggling a bunch of tasks every few milliseconds bothers you, please read on and get more comfortable with the idea. Our analogy will be between the Multi-Tasker and a busy Dentist's office. It takes a bit of imagination, but it should help you get a better idea of how the computer manages the multiple tasks.

Multi-Tasking is much like managing a Dentist's office with several patients in different types of rooms. The patients are at various stages of different dental procedures, some of which are minor and others that are

major. One patient may be in a small treatment room for a filling and is waiting for the anesthetic to take affect. Another may be in the dental tech's room for a cleaning and is waiting for the X-rays to develop. And another is in the surgery room in the process of getting a root canal (ouch!). There is also an administrative area where the Dentist takes care of business and receives instructions on what to do in the various rooms. The office is a fixed size, and the space is divided among the various rooms to suit the needs of the work that is normally done there.

The Dentist is the worker in this case. He devises a system to manage his time and the office resources, but only he can do the work being considered in this example. The goal is to make the most of the dentist's valuable time. The office procedure does this by managing the moment-to-moment operations while simultaneously serving the needs of all the Dentist's patients.

Obviously, the Dentist can only work on a single mouth at any instant. But, he routinely moves from one patient to another so that he is working on several patients during the same period of time. He spends some amount of time on each one before moving on to the next even though the procedure for each patient is not complete. However, the patients feel like they are getting personal service because they expect delays at various stages of their procedures. It would be wasteful (and expensive!) to have the Dentist sit there and tap his shoe while these delays occur. So, he goes off and takes care of each patient one step at a time. Each time period is often called a "unit" in dental terminology, and a typical unit is 10 minutes. Some patients may require more consecutive units than others, but as long as the Doctor eventually gets to each room every everyone is happy.

Now suppose an emergency occurs and one of the patients needs immediate attention. This is an interrupt for the dentist, and he must stop whatever he is doing and run quickly to another room to take care of the problem. Before leaving, he can crank up the nitrous oxide and put the patient to sleep so that he will not notice that the dentist has left. When the emergency is over, he can return to wherever he was and awaken the patient to continue the procedure. The emergency gets handled, and the sleeping patient never knows the difference (unless of course he looks at his watch). This interrupt handling capability is essential in allowing the Dentist to leave a patient while still offering the security of knowing he can be there quickly when he is really needed.

The Dentist has a special duty to perform once per hour. This is when he stops for a moment and returns phone calls from patients with a special problem. The Dentist does not promise to take phone calls at any moment, but he does have a policy of returning calls at the start of every hour. Once he starts the phone session, he calls everyone who called in during the previous hour. This is a top priority task for the Dentist, but it must wait until the appointed time before it is started.

So, the Dentist is a busy guy who takes care of a lot of separate procedures at the same time. He also has to make a living and is always conscious of the needs of the administration area. Of course, only a single patient is actually treated at any one instant. But over a period of time, all patients are attended to in a manner that makes the best use of the Dentist's time and office resources.

The Dentist's environment is *multi-tasking* because he is doing many semi-related tasks at the same time. The priority of each varies according to the needs of the various tasks as well as the resources available at any particular instant. The process of managing the office must allow for many variables such as the needs of the procedures, the speed of the Dentist, the number of patients, the size of the rooms, and the help available from assistants and special equipment. The keys to the success of the process are that no single task needs the *constant attention* of the Dentist, and that the tasks are willing to cooperate with each other in sharing the Dentist's time.

Now for the analogy. The CPU in the multi-tasking system is much like the dentist because there is only one of them that must be shared. The patients in their separate rooms are just like the program's tasks, each with its own set of requirements and problems. The tasks are similar to one another in that they require some effort from the processor, but they each have unique needs of their own. Each task has periods when a delay is appropriate and the CPU is not needed for a while, and like the patients the tasks are willing to give up the CPU's time for these programmed delay periods. Computer memory space is shared among the SCADA tasks just like the limited office space is shared among dental rooms. The CPU divides its time just like the

Dentist does, but is in units that are much smaller than the 10 minute chunks used by the Dentist. The PC operates in *system or "clock" ticks* that occur 18.2 times per second. *This is an extremely inconvenient time period which can be traced to the origins of the IBM PC in Boca Raton, Florida, and is a pet-peeve of SCADAWARE's developers.*

The priority of each patient determines how many consecutive time units he is allowed before the Dentist plans to move on. Likewise, the priority of each SCADA task determines how many consecutive clock ticks a task can have before the MTS forces the processor on to the next waiting task. If the SCADA tasks were not specially programmed for multi-tasking, they would consume all available CPU time even when they had nothing important to do. This would be a waste of computer power and would result in very poor multi-tasking performance. However, SCADA tasks are smart enough (like a dental patient) to voluntarily give up the CPU when they are not very busy. This occurs quite a bit in SCADA applications where a task must wait for relatively slow I/O operations to complete before moving on to the next step. The tasks run for a while, sleep for a while, and then run again all under the management of the MTS. But a task that needs special attention (like the one second SYSTEM task) can be given special priority that allows it to step in line ahead of other waiting tasks. TEST's system uses this technique, called *cooperative scheduling*, which is typical in PC based multi-tasking systems.

The Dentist asks questions and gets responses from the patients as a routine part of his job. He services their requests, one at a time, but can only help them when he is in their room. But the patients are not his only source of information. The Dentist may also get messages from outside sources while he is busy treating a patient. The input will most likely come from the administration area, but it may sometimes come from the lab or the storage area. The information may tell him what to do with the current patient or it may apply to one of those waiting in another room.

To keep his messages straight, the Dentist sets up an in-basket in each room that has a "first-in first-out" design. Messages placed in the basket will be read in the same order that they are received. With this system, the Dentist can go to each room and complete his current operation and then check the in-basket to see if any more work is needed on the patient in that room. The administrators, lab techs, and other support people can stay in touch with the Dentist as he makes his rounds in the office.

The input from the patients is similar to computer user's input from the keyboard or serial port. The Dentist listens to a request, and then preforms a service for *that particular patient.* Input from one patient will not normally affect another one unless a patient *sends a message to another room.* For example, one patient may request the cost of a cleaning and if it is acceptable, he may say "Great. Please clean my child's teeth in the other room." The conversation is between one patient and the Dentist in room one, but the activity will occur in room two. The personal conversation resulted in a message being sent to another room. When the Dentist gets to that room, he will find a message (which he actually sent to himself) telling him the procedure to perform on the patient that is waiting there.

This is somewhat like a local CRT/keyboard user answering a prompt about performing a shutdown operation with the SCADA, but requiring that a message be sent to another task to perform the actual work. Sending the message to the wrong task will not get the desired result. The Dentist's inter-office message system is similar to the one in the MTS because it allows for a first-in first-out queue in the form of inter-task messages. It is also similar in that once the Dentist reads a message, he will perform the complete operation before looking again at the message basket. The MTS does the same thing when a message tells a task to process a command file or library procedure. Command File processing is like a multi-step dental procedure. Once it starts, it cannot be terminated unless the entire task is restarted.

The Dentist's phone call return policy is much like the one-second SYSTEM task in the SCADA program. Any requests for a phone call are stored and processed by the Dentist exactly on the hour. In the SCADA program, any periodic processing requirements are flagged constantly but are processed on each second. So, the SYSTEM task has a special priority just like the phone call routine performed by the Dentist. Once started, the SYSTEM task runs to completion regardless of the number of system ticks needed to make a complete pass.

As more tasks are added to the CPU, the management problems increase. A higher workload will

require additional CPU power in order to prevent a slowdown in overall processing. The multi-tasker is adjustable to suit the exact needs of each individual situation, and it also allows dynamic changes to suit needs that pop up under special circumstances. The multi-tasker allows priority components of any task to execute ahead of routine procedures so that time critical functions can be accommodated. And the MTS takes steps to insure that tasks do not interfere with one another by coordinating the limited resources of the computer such as memory, disk files, and communications ports.

TEST's Multi-Tasker Supervisor is specially designed to work in a SCADA system environment. It allows field configuration of the number of tasks, types of tasks, priorities, programmed delays, and memory workspace. Some of these settings are done in a special file that is processed when the system is first started. Other settings are dynamic and can be changed while the program is running. The end result is a smooth running system that makes the best use of the available memory and processing power for each separate installation.

6.3 SCADA PROGRAM TASK TYPES

There are several types of normally scheduled SCADA system tasks and several special tasks that are constantly being juggled by the tasker. The types of tasks supported by the MTS are:

1. **RTU TASK:** RTU tasks can receive command lines from the operator, other computers, other tasks, or prepared command files. The use of several RTU tasks allows the local user and additional remote users to access a system at the same time. Each user has control over his own task and can operate without much concern for the other users.

On all systems, *task 0* is automatically defined during startup as an RTU type task. This task, referred to as the LOCAL task, is used to process commands from the local user. For this task, commands are usually received from the keyboard and responses are sent to the local terminal. This is the only RTU type task that is automatically defined and started. All other RTU type tasks are optional and will not exist unless explicitly defined by the user in the main configuration (DAT) file.

Additional RTU tasks can be defined to handle the processing of serial and network communications. An additional RTU task is used for each communications channel. For example, if a system communicates only by phone then only one RTU additional task would be defined. However, if a system supports both phone and radio communications, two additional RTU tasks would be defined.

Task 1 is usually set up as the primary communications task for a system. By default, each RTU task is associated with the serial port having the same number as the task itself. For example, task 1 would use COM1, task 2 would use COM2, and so on. If necessary, the comm port associated with a task can be changed by using the SET PORT COMMAND. Refer to the COMMAND REFERENCE MANUAL for more information on this and other available commands.

2. **UTILITY TASK:** The UTIL task is basically just another RTU task with no terminal or serial line associated with it. It receives commands from other tasks or command files and is used for background functions that do not require interaction with the user or other computers. It is defined like any other RTU task and given the ID (nickname) of UTIL in the DAT configuration file. It is told not to use a communications channel by placing the command SET COMM -1 in the task's startup procedure. A few examples of what the UTIL task is used for are as follows:

By default, the UTIL task is responsible for writing log data stored in RAM to a disk based log file. However, the task responsible for this function can be changed by using the LOG TASK command.

The UTIL task also processes the IMAGE save functions, although this can be changed if needed with an IMAGE command option.
The audio playback system also uses the UTIL task by default.

The UTIL task will process report requests if the command LOAD REPORT is processed from within the main configuration (DAT) file during startup. Otherwise, reports will be processed by task 0.

Perform actual processing of command files in response to a request initiated by a new alarm.

3. **SCAN TASK:** This is a background task used to scan all channels for new data, check for alarm conditions, and decrement alarm timers. Upon detection of alarm condition changes, this task is responsible for logging values to memory and activating links for channels configured to call on alarm or reset. This task also initiates execution of command files for channels programmed to do so (READ commands are sent to UTIL task),

   The SCAN task also executes PID and AGA3 calculations as part of the scan for those channels. Timer and Totalizer channels are also updated once per second as part of the priority processing done by the SCAN task.

   AGENDA list processing, log system timing, and other internal housekeeping chores are also done by SCAN.

4. **DRIVER:** The DRIVER task is used on systems with local I/O points to perform the low-level I/O operations for status inputs, status outputs, analog channels, and counter channels. This task will control outputs and convert inputs into digital or value type data. While the DRIVER task is responsible for reading input and output values, other tasks are responsible for processing the data without regard to where it came from. If no I/O hardware is attached the DRIVER task is not needed. *Note: The Driver task will not be operated by the MTS unless an RTU security lock (dongle) is attached to the computer.*

5. **SYSTEM (ONE SECOND TASK):** This task is a special task that gets to run once per second. It takes care of system level time-related functions such as callout delays and retrys, data image saves, and report printing. It also toggles the watchdog monitor output if activated with the MONITOR command.

When the program initially loads, it automatically allocates the Local RTU task, the System Task, and the SCAN task. Any other RTU or Driver tasks must be defined in the DAT file used to describe each system. Refer to section 7.11 of this manual for information about how to define a task.

It is important to understand the concept of the multi-tasker because the power it provides is accompanied by a certain amount of complexity. The main thing to remember is that each of the RTU type tasks is associated with a specific communications line, or local terminal, except for UTIL tasks which do not have any physical I/O device. When commands are processed, it is important that the proper RTU task executes the command. All other tasks perform common support functions that are needed by the RTU tasks.

6.4 TASK SWITCHING

The primary function of the multi-tasker program is to constantly switch from doing one function to another so that the computer appears to be operating many programs at once. Task switching refers to the process of halting one function in its tracks and remembering everything about the state of the task at that moment. The next eligible task is then restarted from the point where it was interrupted. This process continues in a never ending loop that gives all tasks a chance to run, one after another.

SCADAWARE's multi-tasker uses the principle of cooperative switching. This means that the entire program is written in small segments designed to execute one tiny portion of the overall operation. There are thousands of these procedures in SCADAWARE, each taking a small portion of time (measured in micro seconds). The cooperative multi-tasker monitors overall operation of the system and stops execution of a task when its allotted time has run out. The basis for timing is the PC's real time clock, accessed either directly or though DOS software procedures, with a time base of approximately 18 task switches per second.

When each task's time runs out, processing proceeds with the next task in the list which is ready to run. A task may not be ready because it is "sleeping," or executing a programmed delay. This occurs when a task knows in advance that it will not require any processing power for a while. These delays occur during serial data transmission, programmed delays, and other timed activities. The Tasker is smart enough to keep track of these delays so that no processor time is wasted on tasks that have nothing to do. This occurs quite frequently in most I/O intensive systems like SCADA systems that spend a lot of time simply waiting for something to happen.

If a task determines that it has nothing to do, it can give up the processor by initiating a task switch much like the timer tick does. In this way, the next eligible task gets a chance to run a little sooner because it will not have to wait for the next timer tick. Although the ticks occur at 18.2 times per second, this is a long time from the computer's viewpoint. The efficiency of the multi-tasker can be observed when it operates a local terminal and two remote terminals with very little apparent degradation in system performance, even on a lowly 8Mhz PC.

6.5 TASK PRIORITIES

The above "round-robin" description of the task switching is a simplification of what actually happens in the real system. Only the primary procedure for each task is executed in this manner. Besides managing the primary task functions, the multi-tasker also provides priority processing for specific functions within each task. These priority procedures are provided a chance to run on each system tick, and once per second, regardless of which task was active at any instant.

The multi-tasker is therefore operating at several different levels. The main loop handles the main functions of each task, the ones which operate in a cooperative manner. The next level is the system tick level, where each task gets a chance to execute selected routines on each system time interval (18 times per sec). The last level is the one-second level, which occurs every 18 system ticks. The one second priority is adjusted for the 0.2 error inherent in the PC system clock such that over a long measure of time (approximately one minute), the system appears to keep very precise time in spite of the PC's built-in error.

The priority of each task defines how many consecutive system ticks it is allowed to operate before the multi-tasker puts it to sleep. The next eligible task will then pick up where it left off when its time had run out. Typical task priorities are 2 and 3 ticks, with longer ones needed only in very special cases. Note that the task priority can be changed at any time, allowing tasks to increase or decrease their processor allocation on-the-fly. This can be useful for tasks that run infrequently, but can use lots of processor time when they are operating. An example could be a serial communications task that is not used often. Part of the "connect" procedure for that task could be to increase its priority, and the "Bye" procedure could be used to restore it to a low level. However, the high efficiency of SCADAWARE's multi-tasker has eliminated concern over task priority because it wastes very little time on high-priority tasks which actually have very little to do.

The one-tick and one-second procedures for each task will be executed automatically by the multi-tasker as required. Each task has its own needs, and they will be executed in the sequence in which the tasks are defined in the DAT file. Any task switching delays (caused by certain DOS and floppy disk operations) will be tracked so that the proper number of passes is made through the tick and second procedures. This may result in the special procedure executing later than normal, but the MTS will insure that they execute the proper number of times as soon as possible after the delay.

Also, some tasks "go critical" from time to time to avoid problems associated with task switching under the DOS operating system. Specifically, any file operations that require DOS services will make a task critical so that it does not allow other tasks to run until the current operation is completed. This produces slight (10ms to several seconds) delays in other tasks that must wait for the critical task to complete before getting another chance to run. The critical settings are done automatically by the tasks themselves and are not directly controlled by the user.

6.6 TASK DELAYS

Some tasks do not need to run that often, as is the case with the I/O driver task. The delay setting for each task is the number of ticks it will go to sleep after each *complete execution* cycle. This means that when a task's program is completely finished it will automatically go to sleep for at least the specified number of ticks. Other tasks, such as the alarm scanner, may also be similarly adjusted so that they do not use up an unnecessary amount of processor time.

This is different from the time delay caused by pausing a task because its time priority has run out. In this case, a task will be marked as ready to run at the next possible opportunity. When a task completes one cycle and has nothing else to do (as when there are no more keypresses to process), it will go to sleep for the number of ticks specified in the delay setting.

The point of the delay setting is to avoid wasting processor time on tasks that do not need to be run constantly. For example, a Host computer will only get new alarms when an update occurs unless there is local point activity from some other source. It may be desirable to only run the Scan task every few seconds by specifying a delay of 30 to 50 ticks. This will avoid wasting processor time checking point values which rarely change. An even fancier application can adjust the delay of the Alarm Scanner "on the fly" during downloads to allow faster alarm processing and then slow it down when the download is complete.

6.7 TASK MONITORING

The STATUS command (from the menu or command line) can be used to monitor the status of the multi-tasking system. The displays show the various priorities, delays, memory space (stack) conditions, current commands, and other real-time parameters for each task. These displays are not normally used in routine operation but are invaluable during troubleshooting sessions.

The STATUS displays move quickly because they show a "snapshot" of the task condition at the moment a single task is running. In other words, the task painting the display is always running when the snapshot is taken because it must be running in order to generate the display. So, it may be desirable to use another display (connected to a comm port) to monitor the status of Task 0 so that its actions during delays and other off-line periods can be monitored.

```
HOST  SYSTEM STATUS 09:16:26

TSK ID       TASK_NAME     STATE   TICKS LEFT DELY LEFT STACK_SIZE
CUR_STACK
0   0        Local CRT     RUNNING     2    2    2       8000
05BF:6B2C
1   1        Com1 Mdrop RadAsleep      2    2    2    2  8000
05EF:7C84
2   UTIL     Utility RTU   Asleep      4    4    2    2  8000
0607:7C84
3   Scan     Point Scan    Asleep      2    2   30   14  8000
0617:7FAA
4   Sec      System        Asleep      2    2    2    2  8000
061F:7FC6

                [ESC] to STOP       [ENTER] to TOGGLE DISPLAY
```

MULTI-TASKER STATUS DISPLAY

7 SYSTEM CONFIGURATION FILES

The SCADA/RTU program is a complex, multi-tasking program that must be set up for each individual situation. The configuration of each system is primarily performed via two text files that are processed when the program first begins operation. The first file, the main configuration file, is used to define such things as tasks, RTUs, number of channels, and communications media. The second file, the I/O configuration file, is used to define the actual I/O being used. This file is only necessary for systems that contain physical I/O devices.

The configuration files are simple ASCII text files that can be prepared and maintained with any text editor, including the one contained in the RTUMON3 (or RTULITE3) program. Each line begins with a keyword and is followed by optional parameters. The processing of the configuration files follows the same parsing rules as the normal command processor. The important point to remember is that either spaces or commas can be used as parameter separators, but not at the same time. If a comma is present anywhere on the line, then it becomes the separator. Otherwise, spaces are used. If any parameter is to be skipped, commas must be used as the separator. Comments can be placed throughout the configuration files by using the semicolon (;) character. When lines are processed, the parser will ignore semicolons and anything that follows them.

7.1 MAIN CONFIGURATION (DAT) FILE

Upon startup, the RTUMON3 (or RTULITE3) program will automatically look for and process the main configuration file. The default name for the main configuration file is "RTU.DAT". However, another name can (and should) be used by specifying /F=xxxxx on the DOS command line that starts execution of the main program. For example, to start the SCADAWARE program with a configuration file called WC458A.DAT the following line should be used:

RTUMON3 /F=wc458a.dat

If no extension is specified in the file name when using the /F option, the extension ".DAT" is assumed.

The main configuration file is used to specify the following:

1. Number of variables that can be defined for each task.
2. Number of communications links allowed.
3 First-In-First-Out buffering of UART allowed.
4. Types of RTU and I/O driver tasks.
5. Number of call out groups allowed.
6. Library buffer size and maximum number of entries.
7. Agenda buffer size and maximum number of entries.
8. Which overlay units to load and lock in memory (SCADAWARE Lite Only).
9. Default Communications media and baud rates.
10. Number of logical RTUs on the system.
11. Channel assignment for each RTU.

This information is necessary at load time because the system must set aside memory for all of these items. This memory allocation is a one-time operation and on-the-fly reconfiguration cannot be done. So, any changes to the main configuration file requires that the system be restarted so that the file can be processed again with the new information.

7.2 I/O CONFIGURATION FILE

All RTU SCADA systems (not Host systems) require some means of getting switch status and analog value data from the actual field devices into the computer for processing by the SCADA program. The RTUMON3 (and RTULITE3) program supports different types of I/O hardware which are used as the interface between the

program and the physical I/O devices. The supported hardware includes the "Metrabus" family of interface boards manufactured by Metrabyte Corporation, simple relay type input and output cards that are installed directly into the PC I/O bus, and other generic PC based I/O cards.

To inform the program of the type of I/O boards that are being used, a TASK statement must be used in the main configuration (DAT) file to define a driver task and the type of I/O boards it supports. Although the driver TASK statement will automatically get processed during startup, it will only tell the system which type of I/O is being used. The actual configuration of the I/O, such as the number of boards and their addresses, is contained in a separate I/O configuration file. The I/O configuration file for each driver task will automatically get processed whenever the task is started.

The purpose of the I/O configuration file is to inform the program of the type of I/O boards that are being used, how many there are, and their addresses. The specific details regarding the information contained in an I/O configuration file can be found in the separate documentation written for each type of I/O subsystem.

7.3 MSG STATEMENT

The MSG statement can be used in the main configuration file to display text messages during the processing of the file. The remainder of the line after the keyword MSG is displayed at the local console. This can be used to tell the user what is going on to assist in troubleshooting during the processing of the configuration file.

```
MSG Defining 12 RTU Configurations
```

7.4 VARIABLES STATEMENT

During program execution each task can define variables which can not be accessed by other tasks. Variables are useful for such things as temporary calculations or prompting the operator for input and receiving numeric values. To keep the amount of storage space allocated for variables to a minimum, the default number of variables allowed for each task is 0. The VARIABLES statement (can be abbreviated to VAR) must be used in the main configuration file to specify the number of variables that can be defined for a task.

Each VARIABLES line in the configuration file is associated with the most previously defined task. For example, to allocate memory for up to 4 variables for task 1, use the statement VAR 4 after the TASK statement defining task 1 but before the TASK statement for task 2. To set the number of variables for task 0, use the VARIABLES statement before any TASK statement. If the number of variables specified is not in the range 0 - 512, the number of variables available for that task will be set to 12.

Once the program is started, variables can be defined for a task by using either the LOCAL or PUBLIC commands. Once defined, public variables can be used by different command files. However, local variables can only be accessed by the files that declare them or by subsequent files called with the GOSUB command. Upon exit from a particular command file, all local variables defined within that file are automatically deleted. However, all public variables will remain defined. The only way to get rid of publicly defined variables is to explicitly delete them using the RELEASE command.

```
VARIABLES 10 ; Need variables for Choke Control
```

rtu config slide

7.5 LINKS STATEMENT

A link refers to the relationship between a RTU and a specific phone number (or radio call sign). When an attempt to communicate with another system is made, a link is activated which knows what number to call and what task will handle the communications. By default, a system can have up to 4 links. The keyword LINKS followed by a number can be used in the main configuration file to change the maximum number of communication links available.

```
LINKS 12 ; One for each RTU plus 2 pager channels
```

7.6 GROUPS STATEMENT

Call out groups are used to specify a number of communication links that will be activated together. For example, when a particular channel goes into alarm it might be desirable to have several links activated. This can easily be done by creating a group of links to activate and assigning that group to the channel. The default number of groups available is 4. The keyword GROUPS followed by a number can be used in the main configuration file to change the maximum number of call out groups available.

```
GROUPS 6 ; Six different alarm callout combinations
```

7.7 LIBRARY STATEMENT

During program startup memory is allocated for a procedure library. The procedure library is actually just a buffer in memory which gets loaded during run-time with groups of commands called procedures. These procedures are equivalent to command files except that the need to use DOS for command file processing is eliminated. By default, the size of the SCADAWARE library buffer is 32K and the maximum number of procedures that can be loaded in the library is 128 (SCADAWARE Lite defaults to 8K buffer with 64 entries). The LIBRARY statement can be used in the main configuration file to adjust the size of the library buffer and the maximum number of entries allowed. The format of the statement is

```
LIBRARY buffer_k_size max_entries
```

where the buffer size is specified in Kilobytes. The buffer size can range from 2K and 64K. If a number outside of this range is specified the limiting value is used. The maximum number of entries allowed can not be set below 32.

Example: `LIBRARY 32 128 ; 32K buffer with 128 entries`

This statement will cause a 32K library buffer to be allocated when the program is started and allow up to 128 procedures to be loaded in the library.

7.8 AGENDA STATEMENT

An agenda system is provided by the main program to allow processing of commands at specific times of day. These commands are loaded during run-time into an agenda buffer from where they are processed at the specified times. The memory for the agenda buffer is allocated during program startup. By default, the size of the agenda buffer is 4K and the number of commands that can be listed in the agenda system is 100. (SCADAWARE Lite defaults to 1K buffer and 32 commands). The AGENDA statement can be used in the main configuration file to adjust the size of the agenda buffer and the maximum number of commands allowed. The format of the statement is

AGENDA buffer_K_size max_entries

where the buffer size is specified in Kilobytes. The buffer size can range from 1K to 64K. If a number outside of this range is specified the limiting value is used. The maximum number of entries allowed can not be set below 32.

Example: AGENDA 16 200

This statement will cause a 16K agenda buffer to be allocated when the program is started and allow up to 200 commands to be listed in the agenda.

7.9 LOAD STATEMENT

The LOAD statement is used by RTULITE3, but not by RTUMON3. The purpose of this statement is to permanently load parts of the program's executable code from the overlay and lock them in memory. Without doing this, certain program functions could only be processed by task 0. Although this usually presents no problem, it is sometimes desirable to have these commands processed by other tasks. The affected functions are:

DISPLAY	Preprogrammed Data Displays
PROGRAM	Line-by-Line serial port configuration
FORM	Screen oriented display form
REPORT	Automatic printed report generation
SMWATCH	Smart Watch PC Clock read/write

For example, the user may wish to use the PROGRAM command to program channels from a remote location. To do that, the PROGRAM command must be processed by a communications task such as task 1. The statement LOAD PROGRAM must be used in the main configuration file to make the PROGRAM command available for use by all tasks, and not only to task 0.

The keywords DISPLAY, FORM, REPORT, and SMWATCH can also be used with the LOAD statement to allow the corresponding commands to be processed by any task. When the LOAD statement is not used the program requires a smaller amount of memory to run. Each additional keyword used with the LOAD statement increases the amount of memory the program requires. Therefore, the LOAD statement should be used only when one of these commands must be used by a task other than task 0. Several keywords can be listed in a single LOAD statement or several LOAD statements can be used to specify different keywords.

LOAD DISPLAY REPORT ; two options on one line

7.10 COMM STATEMENT

Each communications port to be used by the program must be specified in the main configuration file using a COMM line. This allows the system to allocate memory and hook into the interrupt system for these ports. All modifications made to the normal computer system setup are restored to their original settings when the program terminates.

The COMM line takes a number of parameters:

1. **PORT NUMBER:** This is the comm port number, usually 1 or 2 to indicate the system ports COM1 and COM2. The program will support COM3 and COM4 as well, although a high speed computer is recommended if this many ports will be active at the same time.

2. **BAUD RATE:** This is the default communications rate for this port, typically 1200 for modems and 9600 for direct connections. This can be overridden later with the SET BAUD command.

3. **COMM MEDIA:** This specifies the media that will be connected to the port. Possible values are PHONE, RADIO, DIRECT, MULTIDROP, and Keyboard.

4. **PARITY:** The parity setting of the port must be NONE, EVEN, ODD, MARK, or SPACE. The first letter of the parity word is used (N,E,O,M,S). This setting must match that of the computer to which this comm port will connect.

5. **DATA BITS:** The number of bits in the data word used in communications is set here. Default value is 8.

6. **STOP BITS:** The number of stop bits (bits after the actual data transmission) is set here. Default value is 2.

7. **INPUT BUFFER SIZE:** The communications driver in the SCADA program is interrupt driven. This requires that buffer space be set up for both input and output. This parameter determines how many bytes to set up for the input buffer. Default value is 1100, allowing several lines to be input and waiting for processing by the RTU task.

8. **OUTPUT BUFFER SIZE:** Similar to the input buffer, this buffer holds characters sent from an RTU task and awaiting transmission. Default value is 4096, although values as small as 400 will work in most applications.

9. **PC INTERRUPT LEVEL:** Comm ports 1 and 2 have pre-specified PC interrupt levels of 4 and 3. However, ports 3 and 4 are non-standard and normally use another interrupt level, typically number 7. In the case of ports other than 1 and 2, the interrupt level must be specified here.

10. **PORT BASE ADDRESS:** Comm ports 1 and 2 have pre-specified PC port addresses of $3F8 and $2F8. Other comm ports must have different addresses, and the default for port 3 is $3E8 and port 4 is $2E8. If other than these defaults are used, then the port address must be specified here.

The following lines are examples of how comm ports 1 and 3 can be setup:

```
COMM 1 1200 Radio N 8 2 200 400
COMM 3 1200 Phone N 8 2 200 400 7 $3E8 ; Special Port and Interrupt
```

7.11 FIFO STATEMENT

High performance serial port chips, called buffered UARTS, can perform first-in-first-out (FIFO) buffering of serial data. This is a very advanced process which required exact coordination between the UART device and SCADAWARE's interrupt driven serial drivers. The default setting is to not use FIFO buffering as it is not needed and works only on certain serial chips. However, the user can turn on FIFO buffering by entering FIFO ON after the COM line which defines each serial port.

```
FIFO ON         ; enable uart buffering if available.
```

7.12 TASK STATEMENT

There are three tasks which are automatically defined for each system during startup. These tasks are the LOCAL task, the SCAN task, and the SYSTEM task. All other tasks must be defined in the main configuration (.DAT) file using a TASK statement. The format of the TASK statement is:

```
TASK TaskType TaskName [DriverType] [ControlFileName]
```

There are two keywords that can be used to specify the TaskType. They are:

1. RTU - Task type which processes TSP or other protocol commands

2. DRIVER - Hardware interface for physical I/O

If the keyword RTU is specified as the task type, only the TaskName parameter can be specified. There are no other options for an RTU task. The DriverType and ControlFileName parameters have no effect if specified for an RTU type task.

If the keyword DRIVER is specified as the task type, the DriverType parameter must be specified to inform the program of which type of I/O is to be used. The available keywords for specifying a driver type are:

METRABUS	MetraByte Bus I/O Subsystem
RELAY	Simple PC I/O Port for Status I/O
T100	Test Type 100 Voice RTU
GENERIC	Metrabyte and many other DAS08 type I/O
CTM5	Metrabyte 5 Point Counter Input Card.
DAC02	Metrabyte 2 Point Analog Output Card

When defining a driver task, an additional parameter can be used to specify a particular control file. If the ControlFileName parameter is not specified, the default file names that will be used for each of the drivers listed above are:

RTU.MB	Metrabus
RTU.RLY	Relays
RTU.LB	Type 100
RTU.GIO	Generic I/O
RTU.CTM	Counter Inputs
RTU.D02	Analog Outputs

Normally a specific file name related to the location is used in place of the above defaults. If a control file name is specified, but it does not contain a file extension, the extensions used for the default file names shown above will be assumed dependent on the type of I/O driver specified.

Examples:

TASK, RTU, Com1 Phone
TASK, DRIVER, I/O Driver, METRABUS, WC458.MB
TASK, DRIVER, I/O Driver, RELAY, GA343A

7.13 RTU STATEMENT

Each RTU must be setup individually with an RTU line and corresponding channel assignment lines in the main configuration file. The RTU statement tells the system to start a new RTU using the information provided by the statement parameters. The parameters are:

1 RTU ID: This is a short (up to 8 characters) identification code that will be used to reference the RTU. The name must be a valid DOS file name, meaning it can contain no spaces or special characters. Typical names are VR167, GA343A, etc. This name must be unique in the entire network of computers that will access the data from this RTU.

2. RTU NAME: This is a longer, full name that is used on RTU reports and displays.

3. RTU DEFAULT LINK: This is the default comm link that the program will use to reach this RTU (if it is a remote).

4. REMOTE FLAG: If the keyword REMOTE is at the end of the line, the system will mark this RTU as remote to this computer. This means that logical points associated with this RTU are assumed to be connected to another computer. The data for this logical RTU is expected to come in over some sort of

communications rather than through hardware attached directly to this computer. Local points are updated constantly (via the Driver task), while remote points only get updated when a comm link is established. Sometimes it is convenient to treat a remote point as local and vice versa. In these cases, individual points can be set to remote or local with the CHANGE command at any time the system is running.

7.14 CHANNEL ASSIGNMENTS AND LOG LIST SIZE

The quantity of each channel type must be defined for each RTU (except RTU 0) in the main configuration file. RTU 0, which is the system level, is like a large RTU that contains all the points defined for all other RTUs. In other words, the other RTUs are thought of as subsets that make up RTU 0. Consider, for example, a system that contains 2 RTUs. If 10 status points are defined for one RTU and 20 status points are defined for the other RTU, RTU 0 would automatically contain all 30 status points.

It is good practice to install spare points when initially setting up a system. This makes changes easier because only channel reconfiguration is required to add points in the future. For example, a system with 16 installed analog points may only use 5 points during the initial installation. In this case, it would be reasonable to set up a total of 8 analogs so that 3 spares are readily available without having to edit the DAT file to add one or more new analog points.

The memory allocated for the memory log list can also be specified in this section. Each RTU has a separate pool of memory buffers to hold alarm and data logs. This memory list is normally eventually written out to disk and will not affect the total number of logs that can be stored. The write to disk time is adjustable, but is generally in the range of 10 to 60 seconds. This memory list size will affect how many *concurrent* unwritten logs a single RTU can have. If the list length is exceeded before it is written to disk, then subsequent logs will be discarded until there is room in the buffer. The default list length is 16, which is more than adequate for most systems. The list size can range from 0 to 512. If numbers outside this range is specified, the limiting value will be used. This option is available for special RTUs that will have many simultaneous alarms that must be logged at the same instant.

Channel quantities and log list size for each RTU are defined with the following keywords:

STATUS - Status inputs
AIN - Analog Inputs
PID - Analog Outputs
OUTPUT - Status Outputs
COUNTER - Counter Inputs
TOTAL - Totalizer Channels
TIMER - Timer Channels
AGA - AGA3 Meters
FUNCTION - Rate, Difference, Average, or Special Procedure Channels
VALUE - Simple Real Value Channels
LOG - Size of Alarm and Data Log List

As each RTU is defined, the channel type keywords are used followed by a number which indicates how many points of that type will belong to that RTU. Each RTU, except RTU 0, must have an entry for each channel type that the RTU will access. In the case of RTU 0, the number will be calculated automatically after the total channel count has been determined after the last RTU is completely defined.

A typical setup for a single RTU might be:

```
STATUS      8  ; 8 status points
Output           8
Timer            8
AIN         15  ; 15 analog input points
Counter     3
```

```
AGA3        2
Value       12
Total       2
PID         2 ; used to drive analog outputs
```

As the channel keywords are processed, they apply to the most recently defined RTU. As additional RTUs are defined (see RTU statement), subsequent channel lines will apply to those RTUs. In this way, each RTU can be set up individually.

7.15 NAME STATEMENT

This is very similar to the RTU statement above but relates only to the system level RTU called RTU 0. It is used in the main configuration file to complete the definition of the system RTU (number 0) which is automatically started when the program begins execution. The parameters are the same as the RTU statement. The RTU name becomes the system name for the entire computer. No actual points are needed for this RTU as it will inherit the sum of all points defined for all other RTUs. So, use the NAME statement to complete RTU 0's definition before moving on to the definition of any other logical RTU.

7.16 TAP STATEMENT

TEST SCADA PROTOCOL (TSP) provides for passive monitoring of a communications line to update "listeners" without addressing messages specifically to them. This "wire tap" feature requires system resources which must be allocated during startup. The TAP statement tells the system how many logical RTU names will be monitored by all tasks which are on multi-drop communications lines. The list of names does not take up much memory. However, the processing time required for the TAP process is non-trivial, and the function is deleted from the system code if the TAP statement is not used in the DAT file.

```
TAP 16  ; provide monitoring for 16 rtus
```

Refer to section 21 for more information on Mulit-Drop communications and TAPs.

7.17 EXAMPLE OF MAIN CONFIGURATION (DAT) FILE

The following is a sample configuration file:

```
; RTU setup file for WACKER Galveston 343 A
variables 12  ; Applys to RTU 0 only
links 8            ; Total links in the entire system
groups 5           ; Five separate alarm callout groups

msg Setting Up com ports
COM 1 1200 radio N 8 2 200 400
com 3 2400 phone N 8 1 200 400 7 $3e8 ; Special port

msg Task definition
task, rtu, Com1 Phone             ; task 1
task, rtu, Com2 Radio             ; task 2
task, rtu, Utility Task            ; task 3
task, driver, I/O Driver, generic, ga343.gio ; task 4

msg Finishing RTU 0
name, GA343A, WACKER OIL GA343 RTU
```

```
msg Starting RTU # 1
rtu, GA343A, WACKER OIL GA343A,1   ; start a local RTU
status 8
output 4
ain  16
count 4
total 4
timer 4
aga  2
func  5
value 10
log    32  ; allow for lots of simultaneous logs

msg Starting RTU # 2
rtu, GA351B, HARDY OIL GA-351B,1,r  ; start a remote RTU
status 8
output 4
ain  8
count 2
```

```
total 1
timer 4
aga  1
value 8
 ; ----------- End of sample DAT file ------------------
```

8 TSP COMMAND FILES

8.1 TSP COMMAND FILES

Each RTU type task processes commands that can come from a variety of sources. The most basic source is direct keyboard entry by an operator. However, tasks can also process commands which are stored in text files. These files, known as "command files", contain sequences of commands that perform specific functions. The commands processed from within a command file are processed just as if they were entered manually from the keyboard. A list of all available commands can be found in the TEST SCADA PROTOCOL COMMAND REFERENCE. Once a task begins processing a command file, it can not accept commands from any other source until file processing is complete.

Command files can be prepared with any text editor or word processor that generates simple ASCII text files. Editors that generate special word processing codes can not be used because the special codes will confuse the program. The SCADA program itself contains a simple text editor that can be used by issuing the EDIT command once the program is started.

Command files can have any name and file type, although a specific file type is expected for certain situations, such as RTU task startups and communication events. Other command files can have any valid DOS filename because they will be processed only on command.

For example, each task starts execution by looking for a file called STARTx.RTU, where the x is replaced with the task's number. When the local task starts up, it looks for START0.RTU. Task 1 looks for START1.RTU, and so on for all RTU type tasks.

Similarly, the communications events look for command files called LINKx.RTU, CONNECTx.RTU, BYEx.RTU, and COMFAILx.RTU for their respective functions. If the files exist, they are processed. If they do not, then nothing is processed as an alternative.

The contents of these files depends greatly on the particular installation. The general functions are the same for most applications, but the exact commands and their parameters cannot be easily duplicated for each installation because of differences in hardware and communications channel requirements. Some of the standard command files used on most systems are explained later in this manual beginning with section 8.5.

8.2 COMMAND PROCEDURE LIBRARY

It is possible (and desirable) to combine several command files into a single file and define each group of commands using the line PROCedure *procname*. This procedure file can then be loaded into a buffer in memory which will act as a library of command files. This eliminates the need to use DOS for most command file processing because the text of the file is already in memory. It also simplifies maintenance because many files are combined into one library file and there are fewer files per RTU.

When the program processes commands the library will be searched first before looking on disk for command files. This avoids disk accesses for common procedures. The search sequence is to check the library, then the disk for a file with an .RTU extension. However, If the READ or GOSUB commands reference a file with a file type specified, then the library search is skipped. This is illustrated by the following commands.

```
READ START1    ; check library and if not found then disk
READ START1.RTU  ; go directly to the disk bypassing library
```

This is important because any existing operations that specify a file type in the READ command line will have to be modified in order to benefit from the library function. Also, files that are normally processed from disk (like setup and Link files) will be accessed faster if their file types are included on the command lines that read them.

The LIBRARY command is used to load files into the procedure library in memory. By default, SCADAWARE sets up a library buffer of 32K with room for 128 procedures. SCADAWARE Lite defaults to 8K and 64 procedures. The LIBRARY command can be used in the main configuration (DAT) file to adjust the size of the library buffer and the maximum number of entries allowed. The format of the DAT file setup command is

LIBRARY buffer_k_size max_entries

where the buffer size is specified in Kilobytes. The buffer size can range from 2K and 64K. If a number outside of this range is specified, the limiting value is used. The maximum number of entries allowed can not be set below 32.

For example, the following line could be used in the main configuration file to change the size of the library buffer and the number of entries allowed:

LIBRARY 20 120 ; set up 20K buffer and allow 120 procedure entries

An example of a small library file is as follows:

```
procedure start1  ; Startup serial port task number 1
  set port 2      ; override default selection of port 1
  set media phone
  set wait 10     ; time to wait for an ACK from RTU
  set trys 4   ; number of callout attempts
  set priority 3  ; 3 ticks for this busy task
  force local echo Task 1 started at $T $D

proc bye1
  SET ECHO OFF
  hayes S0=1 Q1 E0

proc daily  ; execute this once per day
  sele wc640
  calc v4 = v5
  calc q1:q3 = 0

; end of library
```

When this file is loaded with the LIBRARY LOAD command, it will make the procedures START1, BYE1, and DAILY available quickly and without DOS accesses. Note that either the keyword PROCEDURE or PROC can be used to indicate the beginning of a new file in the library.

If a new procedure with the same name as an existing procedure is added to a library via either the LIBRARY LOAD or LIBRARY MERGE command, then the new one replaces the older one. The old one is still there, it is just de-activated. It will appear in the DUMP of the library with the first letter of the procedure name replaced with a question mark (?).

8.3 COMMAND FILE PARAMETER PASSING

Command files and library procedures are read with parameters passed in the command line. This is similar to the parameter passing in DOS batch files. The intent is to allow for similar operations that occur in slightly different ways to be processed by a common file. *This is an important concept that can greatly simplify command file preparation and enhance the capability of the system*. If you are new to programming you may have to read this section more than once to let it all soak in.

Parameters are any text items on the command line that follow the file name in the READ or GOSUB

command line. Each parameter is referenced by a number, with parameter number 1 being the first item after the file name. The file name itself can be referenced as parameter 0, although this is rarely needed in actual practice. The parameters are referenced within the command file as special $ codes, similar to the special codes in the ECHO command. For example, the second parameter on the command line is referenced as $2 within the file.

What this means is that a standard command file can be written that uses "place holders" rather than exact text that is needed to perform a certain function. These place holders, called the prototype parameters, are always indicated with a dollar sign code from $0 through $9. The file is written with the prototypes in place of the values that will be substituted at runtime. The runtime parameters will hold the actual text that is needed to execute the desired function.

This may sound confusing but it really is not difficult to understand, especially after a few examples. Think of a command file that uses parameters as a paper form with a few blanks that need to be filled in. The concept of runtime parameter substitution is to build a command file that acts much like the blank form. Instead of using "blanks" to mark spaces that will later be filled in, the command file uses the prototype parameters marked with the $ codes. The file contains most of what is needed to perform the final act with only the items that will change being left "blank". Lets create a form that will let us specify a name, age, and occupation for different people. Most of the form is the same for each person, with a few changes needed to indicate the different data.

Name: _____
Age: _____
Job: _____

This is easy enough to grasp, right? If you were filling in the form, you would need three pieces of information (name, age, job). A command file is very similar in that some information is constant while other parts are variable. A pseudo command file might look like this:

```
Name: $1
Age:  $2
Job:  $3
```

Note now the $ codes take the place of the blanks in the earlier example. Now the trick will be to get the needed information into the right place. The way we do this is to "pass" the parameters to the form (the command file) in the proper order. *The order of the parameters is extremely important and should be standardized in all your applications.* If our form was a file named "INFO" then we could "run" the form with the line:

```
READ INFO Arthur 26 Engineer
```

Here, the parameters are expected to be in the order of Name, Age, and Job. In reality, SCADA programs aren't forms but are sequences of instructions that will be executed at runtime. We all know that Arthur isn't 26 either, but this is just an example. Anyway, if we could run this fictional form with the above command line it would look like this *at runtime:*

```
Name:  Arthur
Age:  26
Job:  Engineer
```

Note how the prototype parameters $1, $2, and $3 were *replaced* by the runtime values of Arthur, 26, and Engineer. If you can follow this clearly, then you are ready to look at a real RTU command file. If not, then make another pass through this section until it makes you raise your finger and go "A-ha! Now I got it!"

Lets look at a more realistic SCADA file application. The line:

READ TESTFILE WC45 O1 ON

is parsed as follows:

PARAM	*VALUE*
0	TESTFILE
1	WC456
2	O1
3	ON

A command file that is designed to call an RTU and set an output off or on might look like this:

```
; Param 1 is RTU, param 2 is Output, Param 3 is off/on condition
sele $1           ; Rtu name
dial
wait 20 conn
set online on
msg RTU $1 is online
block calc $2 = $3  ; Calc channel to a value
msg Output $2 is now $3
hangup
```

As each line is read from the file, the file processor substitutes each $ parameter with the corresponding entry on the line. So, the translated file would look like:

```
sele WC456
dial
wait 20 conn
set online on
msg RTU WC456 is online
block calc O1 = ON
msg Output O1 is now ON
hangup
```

Notice how each $ entry was replaced automatically by the command processor so that the file appears to be prepared especially for this particular operation. The same file could be used for any output at any RTU, in either OFF or ON mode by entering the proper command line. Similar files could be used for pulsing timers, scanning Hi/Lo set points, etc. Now keep in mind that it is possible to write a complete, separate command file for every possible action that will be required of the SCADA system. But isn't it easier to design things so that a fewer number of "generic" files can be used to do similar tasks?

Normally, these command lines can be set up in a menu so that the operator need only select the desired operation. Each menu entry would be similar, with only the parameters changing in each line. Any changes needed to the common file would only have to be made once, eliminating redundant files for different RTUs. Consider this small section of a menu control file:

```
|Perform WC-240 Facility Shutdown
  Read SHUTIN WC240 T5 45
|Perform WC-240 Compressor Shutdown
  Read SHUTIN WC240 T6 30
|Perform HI-121 Well Shutin
  Read SHUTIN HI121 T2 60
|Perform EI-232 Facility ESD
  Read SHUTIN EI232 O3 ON
```

These menu entries will offer 4 separate suggestions from which an operator can select an output control action. Each selection will cause an *almost identical* action to occur. The same file will be processed but the command line parameters will be different. This allows a single file, *SHUTIN*, to be used for all four functions because the differences are specified by the menu command line. The end result is that it is much easier to write, test, and debug your command files because a fewer number of them are needed to perform a wide variety of functions. The menu resulting from the above file would look like this:

Perform WC-240 Facility Shutdown Perform WC-240 Compressor Shutdown Perform HI-121 Well Shutin Perform EI-232 Facility ESD

8.4 CMD FILES ON ABNORMAL/NORMAL CONDITIONS

Each channel has a mode setting option which, if set, will allow a command file to be executed each time the channel enters an abnormal state. The name of the command file must be the same as the channel tag and have the extension ".RTU". This option can be set for each channel by using the CONFIG or PROGRAM command. The prompt for this option is "Execute RTU File?". All value type channels also have the option of specifying an Input Control channel. If one is given, the command file execution will be disabled as long as the Input Control channel is in an abnormal state.

Each channel can also execute the same command file each time the channel returns to normal. Two conditions must be met to enable this feature. First, the option just mentioned, which allows command file execution on abnormal, must be set. Second, the global variable RFILE must be set. To set this variable use the command SET RFILE ON. Unlike the setting for file execution on abnormal which is a separate option for each individual channel, this setting controls the file execution on return to normal for all channels on the system.

When a command file gets executed the program passes a single command line parameter which depends on the state of the channel. If the channel has entered an abnormal state the value 1 is passed as a parameter to the command file. The value 0 is passed when file execution is caused by a return to normal condition.

For example, a channel tagged HORN will cause the file HORN.RTU to be executed by the UTILITY task whenever its state changes. The actual line sent to the UTILITY task will look like READ HORN 1 when the channel goes abnormal and will look like READ HORN 0 when the channel returns to normal.

All command files should contain simple logic statements to determine if the file is being processed as a result of a new abnormal or a clearing one. This is easily done as shown in the sample command file below:

```
if $1 = 1
msg This is a new abnormal condition.
; place commands for abnormal condition here
else
msg This is an abnormal condition clearing back to normal
; place command related to return to normal here
endif
```

8.5 STARTx.RTU - TASK STARTUP FILES

When all RTU type tasks are started, they automatically process files called STARTx.RTU, where the x is replaced with the corresponding task number (0,1,2,..). When the program begins, only task 0 (the local user) is automatically started and the file START0.RTU is automatically processed. *All other tasks must be started by task 0 once it is running.* Therefore, START0.RTU is normally used to setup any system wide controls and

start the other tasks.

The general sequence of events in a START0 file are as follows:

1. Load procedure library
2. Load existing Data Image from disk file

-or-

2a Load RTU channel files.
2b. Load agenda list.
2c Load link information.
3 Set special channels or variables to pre-set conditions
4. Start other tasks.
5. Load Agenda List
6. Setup Logger Data Saves
7. Start Image save
8. Put up main menu or user menu.

The most common means of accomplishing these steps if to use a generic START0.RTU file which contains the following information:

```
LIB LOAD $S.LIB   ; load in the default library
read startup            ; branch to standard procedure name
```

This simple file can be used on any system provided that 1) its main library name is the same as the overall system name, and 2) a procedure named STARTUP exists in the library to complete the initialization process.

8.6 CONNECTx.RTU - TASK CONNECT FILES

When a system connects with another unit, either from making a call or receiving a call, it will look for and process a file called CONNECTx.RTU if it is found. The x in the file name is replaced with the number of the task handling the communications. These files can be used to do anything that must be done when a connection is made. For example, a CONNECT file can be used to do something as simple as count the number of times a connection has been made.

8.7 LINKx.RTU - CALLOUT CONNECTION

When a task on one unit attempts to make a connection with another unit it will wait for a serial port signal called "DCD" before proceeding with file execution. On any DCD the task will execute the CONNECT file mentioned above. If the task was making a call, rather than receiving a call, it will look for and execute a file called LINKx.RTU if it is found. If both a LINK file and a CONNECT file exist, the CONNECT file will get processed first. The x in the LINKx.RTU file name is replaced with the number of the activated link. The LINK file is normally used to change the standard download process of a unit by allowing the unit that originates the call to control the sequence.

On most systems a LINK file is not needed. When a unit connects with another unit and no LINK file exists a default command will be executed. The default command will vary depending on whether the link that made the call out is setup for a Host unit or a RTU. This setting is configurable for each link by using the CONFIG or PROGRAM command.

If no LINK file exists and the activated link is setup for a Host unit, the command BLOCK READ DOWNLOAD will automatically get processed when the connection is made. This will cause a file called DOWNLOAD.RTU to be read at the RTU to have data sent back to the Host. A LINK file could be used at the Host to have it read a file locally and request data from the RTU rather than have the file at the RTU determine what will be sent to the Host.

If no LINK file exists and the activated link is setup for a RTU, the command READ DOWNLOAD will automatically get processed when the connection is made. Again, this will cause a file called DOWNLOAD.RTU to be read at the RTU to have data sent to the Host.

8.8 BYEx.RTU - DISCONNECT FILES

When a communications link is disconnected a file called BYEx.RTU is processed, where the x is replaced with the task number handling the communications. These files are normally used to reprogram the communications channel to make sure that the modem or whatever is properly configured for another call.

All RTU type tasks, except for task 0, are monitored during periods of inactivity. No activity for a certain amount of time will cause a task to process its BYE file. The amount of time to wait for some type of activity before processing the BYE file can be controlled with the SET COMM command. If the timeout period is set to 0 for a task, its BYE file will never get processed automatically as the result of the timeout feature.

Whenever one of the serial communications tasks is started a check is done to see that it is associated with a valid communications channel. If it is, the BYE file for that task is automatically processed.

8.9 COMFAILx.RTU - COMMUNICATIONS FAILURE

If a call out attempt is unsuccessful the program will terminate the call, wait a while, and then attempt to call out again. If this process continues until the number of call out attempts has reached its limit the link will be set to the failed state and a file called COMFAILx.RTU will be processed. In this case the x is replaced with the number of the failed link. This file can be used to sound a horn or whatever is necessary to react to the communications failure. One possible use is to reset the callout system and then initiate a poll, which will basically restart the callout process all over again.

For systems that must communicate with many RTUs, there may exist more than 9 links. When a link with a number greater than 9 fails, a file called CFAILx.RTU will be processed instead of COMFAILx.RTU. For example, if link 9 fails, a file called COMFAIL9.RTU will be processed. However, if link 10 fails, a file called CFAIL10.RTU will be processed. The reason for this difference is to allow a unique file name to exist for each link without exceeding the maximum of 8 characters for any file name.

8.10 LOGERROR.RTU - PROBLEMS IN THE LOGGING SYSTEM

If an error occurs with the logging system when purging the memory buffer, the writing will be stopped to prevent further problems and the UTILITY task will process a special file called LOGERROR.RTU. When the error occurs, a message is sent to the UTIL task (or other user determined task) to process READ LOGERROR. The LOGERROR procedure can be in the library, or it can be in a separate DOS file just like any other TSP procedure. The LOG TASK command can be used to specify a task other than UTIL which will receive the message to process the LOGERROR file.

A common causes of logging error is a full disk device, or a failed printer. The LOGERROR file can do whatever is necessary to change log files to another disk, alert an operator, or simply clear up some disk space by deleting files. The LOG EVERY xx command can be used within the file to restart the logging process after freeing up some disk space. To alert the operator, a channel can be setup (such as a spare status channel not associated with any real I/O) and forced into alarm from within the LOGERROR file. For more information about the logging system refer to Section 13 of this manual.

8.11 LOGONx.RTU - PASSWORD PROTECTED ACCESSES

When passwords are enabled, this file is processed whenever a user signs on with the LOGON command. This allows for special processing such as logging messages of who is using the system (with the LOG MSG and the $U parameter). Each task has its own file, so it is possible to control the action of local accesses separately from remote accesses. The x in the file name LOGONx.RTU is replaced with the number of the task that processes the Logon command.

8.12 "UP" FILES - PROCESSED AFTER DATA TRANSFERS

At the end of a download from one unit to another a SCAN command with an @ sign or the @ENDS keyword is usually processed to get a time-stamp for the time of the update. When an @ or @ENDS is processed in the resulting DATA line by the receiving unit, a command is queued up to read a file with the extension "UP". The name of the file is the same as the ID of the RTU for which the DATA command is processed. The main purpose of an "UP" file is to have the newly received data transferred from the actual channels into a database. This provides an automatic way of continuously updating a database after each data transfer. However, the "UP" file can be used to process any commands the user wishes to have processed at the end of a data transfer.

When a command to read the "UP" file is queued up an UPFILE flag is set for that RTU. Another command to read the "UP" file will not get queued up again for that RTU until the UPFILE flag has been cleared. This will prevent more than one command from being queued up when using a direct link and doing a continuous loop within a file to download data and do a time-stamp. When the "UP" file finally gets processed, the UPFILE flag should be cleared to allow the command to be queued up again in the future. An easy way to always clear the flag when necessary is to have the command SET UPFILE OFF as the last command in the "UP" file. If no "UP" file exists for an RTU then the command to read the "UP" file is simply ignored when it is processed. The SET UPFILE command can be used without any parameters to check the status of the UPFILE flag for a RTU.

```
; Sample Update File for remote RTU. Executes after download is complete
  db sele WELLPRES  ;  select the well pressure database
  db locate $T $D      ; position to current time and date record
  db update       ; let Dbase engine retrieve realtime values
  db close
  set Upfile OFF   ; clear flag so that subsequent Updates will occur
```

9 PHYSICAL & VIRTUAL CHANNEL PROGRAMMING

9.1 CHANNEL DESIGNATIONS

TEST's SCADA system operates on the concept of data "channels" to handle most of the information within the system. Unlike the registers in simple RTU or PLC systems, each channel is much more than a simple binary or numeric value. It is a complete data point within the system consisting of a value integrated with name, tag, setpoints, and other technical features. There are several different types of channels to take care of different types of data. All channel types have some basic things in common, and they also have special features that make them different from the other types. Channels may refer to an actual "physical" data point or to a "virtual" calculated data point like a timer or AGA3 gas meter. This section will provide an in-depth look at all of the channel types and how they are programmed and used within the SCADA system.

The most basic feature of all channels is a way to uniquely reference it within the overall system. There are actually several ways to identify each channel, and the simplest is by a channel ID reference. All channel ID references in the system begin with a single letter indicating the type of channel. The letter is followed by a number indicating the exact channel position in a list for each RTU. For example, *S16* references Status channel 16 for a particular RTU. Channels are numbered from 1 up to the maximum specified when the system was initialized. The type of channel is specified by the letter as shown in the table below.

CODE	CHANNEL TYPE	EXISTENCE
A	Analog Input channel	Physical
C	Counter input channel	Physical
F	Function (rate) channel	Virtual
M	AGA3 Meter Channel	Virtual
O	Output control point	Physical
P	PID Calculation/Analog Out	Physical & Virtual
S	Status Input point	Physical
T	Timer channel	Virtual
Q	Totalizer Channel	Virtual
V	Value Channel	Virtual

Note: The PID channel is detailed in a separate section of this document.

Some of the channels represent actual Physical inputs, while others are Virtual channels that exist only within the computer program. The physical channels represent actual inputs and outputs that are connected to the field devices. The virtuals represent calculated values, timers, and other "intangible" channel types.

The SCADA program groups points of different types into a collection called a "logical" RTU. Each RTU can be treated as a unit, and all channel IDs must be unique within that RTU. So, there can only be one channel T5 in each RTU, but different RTUs can each have their own channel known as T5. A simple channel ID such as T5 is called an *ambiguous* reference because it could possibly refer to more than one channel in the overall system (but only one channel in a single RTU).

The SCADA program always maintains a current RTU that is assumed to be the default location of any channel references. Any IDs that do not specify a particular RTU will be assumed to exist "internally" within the currently selected RTU. Channels that are outside the scope of the current RTU's range are referred to as "external". For example, if a meter in RTU 3 needs to use channel information from RTU 1, then this is an external reference from RTU 3 to RTU 1. To reference an external channel a standard "dot" notation is used. The dot notation consists of two parts separated by a period (or "dot"). The first part is the RTU name and the second part is the channel identifier. For example, VR167A.S3 references Status channel 3 at RTU VR167A. If no dot notation is used, then the current RTU is assumed by the system. If a dot is used, then the system lets the RTU identification override the default selection of the current RTU for that particular reference only.

Note that dot notation can also be used to specify the RTU name even if it is the same as the current RTU. In other words, it doesn't hurt to specify the RTU even when it is already the current one. The system often

does this automatically to insure an *unambiguous* reference to a particular channel. This insures that a system with more than one RTU will not get confused and mistake one RTU's channel A12 with another RTU's A12. If only an A12 reference is used (for analog channel number 12), then it is possible the wrong logical RTU will provide or receive the information that was intended for another unit. By always providing the RTU name as well as the channel ID, any system processing a channel reference will be sure to locate the proper data point within the overall system.

This is not as difficult as it sounds because it is fairly easy to get various computers "in sync" so that they are currently addressing the same logical RTU. But whenever possible the system will automatically append the RTU name for you just to make sure that no chance of a mistake occurs. So, while you may request information on channel A5, the system may respond with WC240.A5 to indicate that the point in question belonged to an RTU named WC240. Don't despair if this is not clear at this point because the concept will be covered several more times before the chapter is through.

9.2 CHANNEL RANGES

Channels can be referenced one at a time or as a group (or range). A range of channels must often be specified for certain commands (like SCAN). The channel range is noted in a manner similar to the cell ranges in a spreadsheet. The starting and ending channels are written together separated by either a colon (as in Microsoft style) or double dots (as in Lotus style). For example, the notation S1:S20 represents a range of channels from S1 through S20. A1..A3 represents channels A1 through A3. The colon or double dot can be used at any time.

For all channel ranges, both the starting and ending channel types must be the same. The program will normally use the channel type of the starting entry and will ignore the channel type of the ending entry. For example, specifying DISPLAY S1..A16 will be interpreted as DISPLAY S1..S16. Wherever possible, the system will check for out-of-range references and chop them to the proper size in any responses. For example, if a system with only 10 analogs was given the command SCAN A1:a16 E, the system would respond with SCAN RTUNAME.A1:A10 E to indicate that only 10 channels were available. Note that the system also appended the RTU name to the beginning of the range to form an unambiguous system level reference to the data channels.

9.3 INITIAL SYSTEM CONFIGURATION

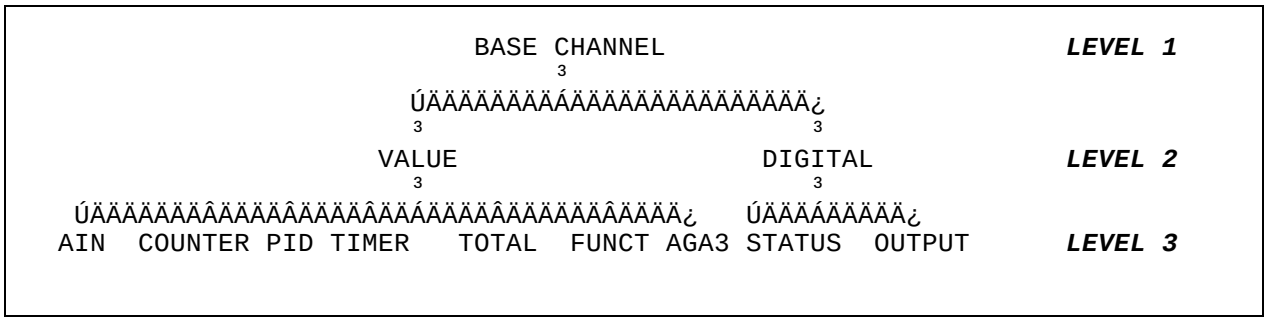
The system is configured for the number and type of channels at the time the program is first started by information contained in the configuration file, which defaults to RTU.DAT. More likely, the default name will be overridden with a file name that is the same as the desired system name such as WC311.DAT or HOST.DAT. If this is done, the /F= command line option must be used when starting the program to tell the system the actual name of the file to be used for system configuration.

The DAT file tells the program how many of each type channel each logical RTU will have. All other channel configuration is done with the CONFIG or PROGRAM command. These commands require a channel number as a parameter, and will enter a prompting sequence to allow entry of channel configuration data. The types of input required depends on the type of channel being programmed. Once channels are programmed, a SAVE command is used to put the channel information into a separate DOS disk file for each logical RTU. So, each RTU ends up with a separate file containing all of the channel information necessary to describe the RTU. This file can be easily transported (by diskette) to another computer to simplify setting up other computers that will access data from the RTU.

When moving a configuration file from one unit to another, there is often a few settings that will be different on the other computers. The most likely change is in the callout requirements for each channel. Normally, an RTU is set to call while a Host is not. The CHANGE command can be used to modify the alarm mode setting for a group of channels. This simplifies transferring a channel setup from a configured RTU to a matching HOST. In these cases, the alarm mode settings are the only differences in the setup. The CHANGE command can also be used in a command file to change the alarm mode settings on-the-fly if needed.

9.4 BASE CHANNEL CONFIGURATION

The TEST SCADA system works internally with an "object oriented" data system where many data types are derivatives of another type. This object orientation is visible to the user when the layout of the various channel types is studied. The system defines a base channel type that forms the basis of all other channel types. This base type has features that are common to all of the other types, although the base type is never used by itself. All channels used by actual data points are more complex than the base type. Any additional requirements for a specific type of channel are processed at a "lower level" than at the base level, but each lower level "inherits" the characteristics of the higher type. This forms a sort of tree structure as shown below:



Internally, the system works with the channels at all 3 levels. Externally however, the user only accesses data at levels 2 and 3. The base channel type has the following attributes that are shared by all other channel types:

1. Channel Name
2. Channel Tag.
3. Alarm Mode.
4. Alarm Delay.
5. Callout Group.
7. Time and Date of Last Abnormal (internally maintained)

All channels have descriptive text messages describing the nature of the channel. Typical values are "Oil Pressure" or "Sales Gas Static Pressure".

All channels also have an "alarm mode" setting that determines how the channel will react to abnormal or alarm conditions. First of all, each channel has the option of being declared as an alarm type channel or not. Each channel also has the option of being skipped or included in reports. This allows for spare and other non-essential points to be skipped during report generation. Each channel can control its ability to blow a horn (only for alarm type channels), execute an RTU file upon abnormal (this also allows execution of an RTU file when an abnormal clears if SET RFILES ON is used), generate a callout, and be entered into an alarm log on abnormal conditions. Each channel can also control its ability to generate a callout on reset and to reset its alarm condition automatically should the abnormal condition go away. These settings can be set up by either the Config command, Program command, or the Change command (for one or a range of channels).

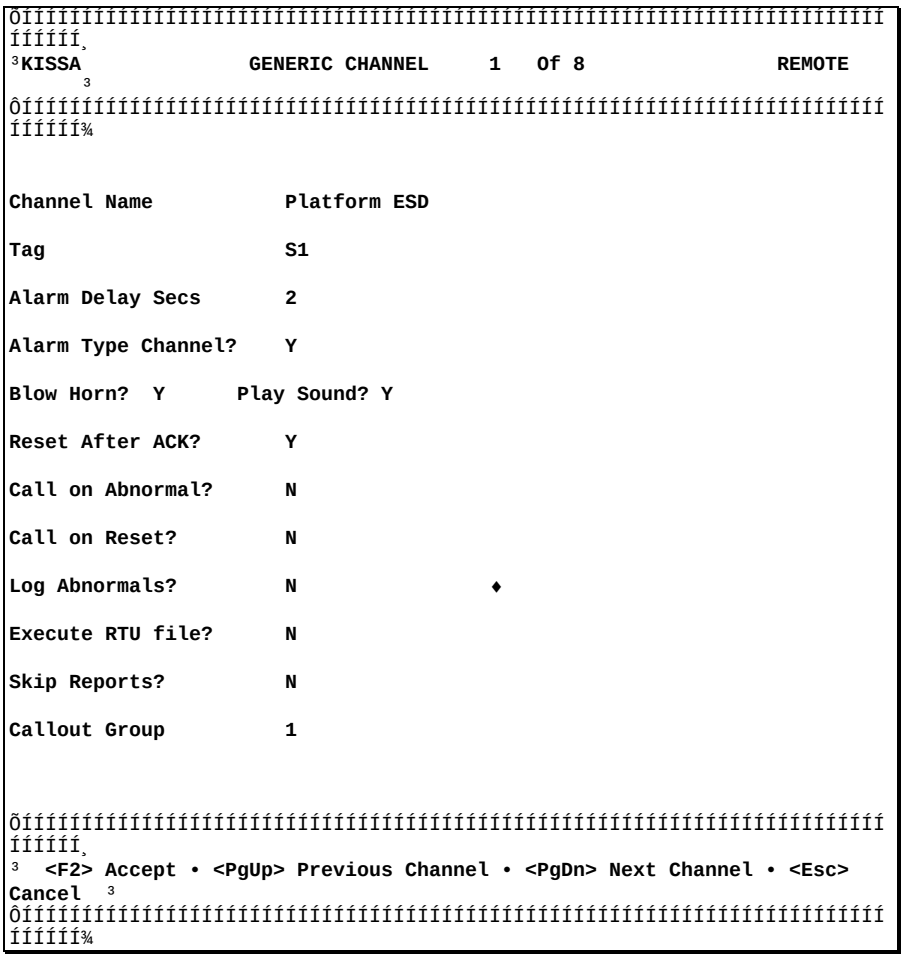
Each channel has an alarm delay setting which determines how long a channel must be in an abnormal condition before going into alarm. If a channel in an alarm condition returns to its normal state before the delay time has expired, no alarm will result. Note that this delay also affects other abnormal actions such as command file processing and abnormal logging.

Channels that cause callouts must have a "callout group" assigned to them. A callout group is a list of communication links that will be activated whenever the point requires a callout. This allows for each point to cause specific callout functions that are unique to its individual situation. See the section on communications links for further information.

Note that each individual channel has *all* of these attributes all to itself. Items like call on alarm, delays,

and other features can be set individually on a point-by-point basis. While this may seem like a lot of setup for each RTU, it can actually be done very quickly with the Change command that can modify a range of channels. An experienced user can perform the base level setup for a typical RTU in about 30 minutes.

Once the base level programming is complete, the system must be configured for the lower level details of each channel. On an actual RTU (one with field devices), all parameters for all channels must be entered. At a Host system (one without actual I/O devices), some of the lower level settings are not important because they will not be used at the Host. For example, an analog channel that must be configured for the Zero and Span of the transmitter need not be programmed at the Host. This is because the conversion from raw to engineering level units (done by the RTU's Driver Task) will usually take place only at the field located RTU. The data from the RTU to the HOST will be passed at a higher level (level 2) so that the details of the transmitter are not important at the Host. This greatly simplifies installation and operation because low level changes at the RTU are not necessary at the Host system.



The channel programming unique to each channel type will be covered in the sections below. Keep in mind that these channels also have the attributes discussed in this section because they inherit all of the capabilities of the base channel. *For each parameter a number will be provided in parentheses to indicate the level at which the parameter takes affect.*

9.5 STATUS INPUT AND OUTPUT POINT PROGRAMMING

Status inputs and outputs are the easiest channel types to understand and program because they represent only off-on type conditions. All that is needed in addition to the inherited base channel parameters is the text messages for normal and abnormal states, and the normal state of the input and output.

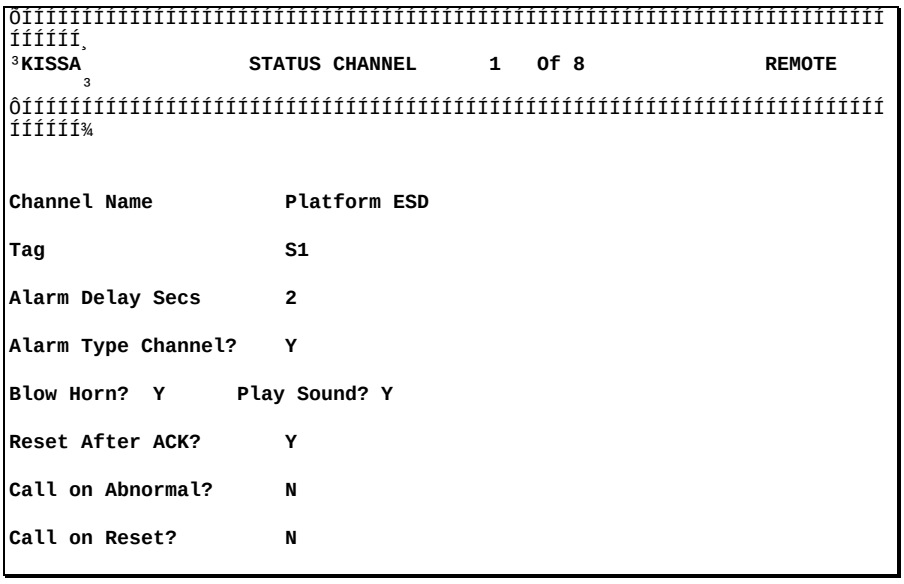
Normal Text (2)	Text associated with the normal state.
Abnormal Text (2)	Text associated with the abnormal state.
Switch Normally On (3)	Normally open or closed state for the input.
Currently Abnormal (3)	Power state of output when logically Off (outputs only).

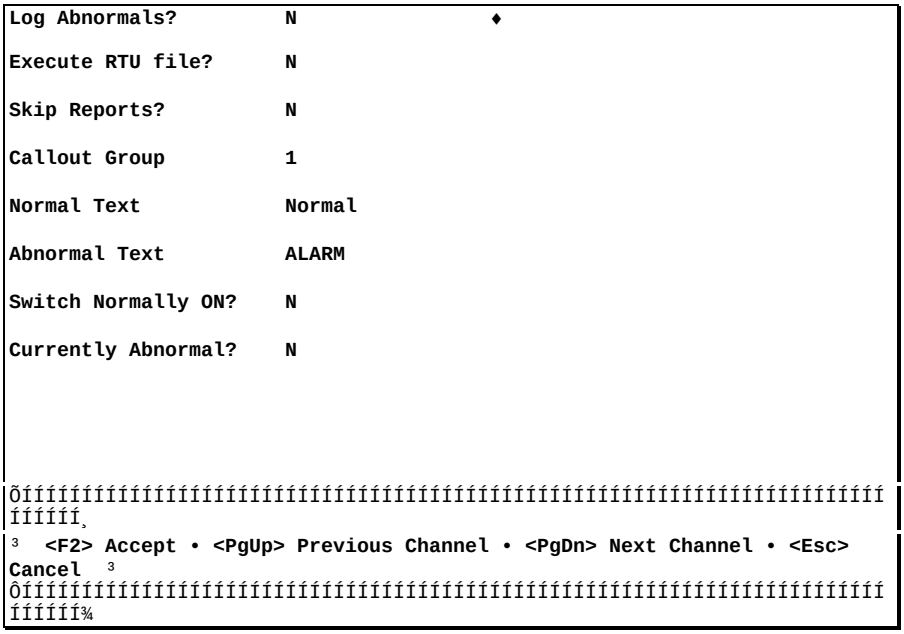
The status channels can only be off or on and have a text message that corresponds to each condition. For example, a status input associated with a pump may have the text messages "Offline" and "Pump Running". The text for each position is completely up to the user. The normally open or normally closed position of the physical input is hidden from the user because each point can be programmed for either state as "normal."

The conversion form the field switch to the logical off/on state is done at the RTU only. Once the switch is read, the driver task makes the necessary adjustment for normally open or closed so that *an abnormal condition is always a 1* inside the system. Therefore, scanning data from one unit to another will always result in a value of 1 for each channel that is abnormal, regardless of the actual field switch position. This reduces problems in keeping the field and host systems in exact sync as far as field switches are concerned.

Likewise, status outputs can also have a normal state of Off or On. This allows all outputs to be turned on by setting them to 1, and off by setting them to 0. The actual needs of the physical output is taken care of by the driver task. Therefore, an output that must be physically On in order to turn Off a function can be setup so that it is always turned off by setting it to 0. For example, an output that is normally energized to keep a piece of equipment operating can be programmed for its normal state to be On. To activate the "function" of the output channel we always set it to 1. The driver will recognize that this particular point uses reverse logic and will turn the physical output off when the logical output is on.

Note that status channels, like any other channel, do not have to be associated with actual field devices. It is often convenient to use status channels as indicators or logic controls within the system. We can set status inputs or outputs off or on and use them as logic flow controls within programs. The off/on status can be tested with an IF statement that will determine various actions in a program. Using status channels allows these various settings to be easily transferred among units. So, keep your mind open for creative applications for status channels as you learn more about the system.





9.6 VALUE CHANNEL PROGRAMMING

Value channels (which are at level 2) require a number of parameters in addition to those inherited from the base channel type. All parameters associated with value channels are also inherited by analog, counter, timer, totalizer, function, PID, and AGA meter channels. *All Value channel parameters are at level 2.* The additional parameters used by value channels are:

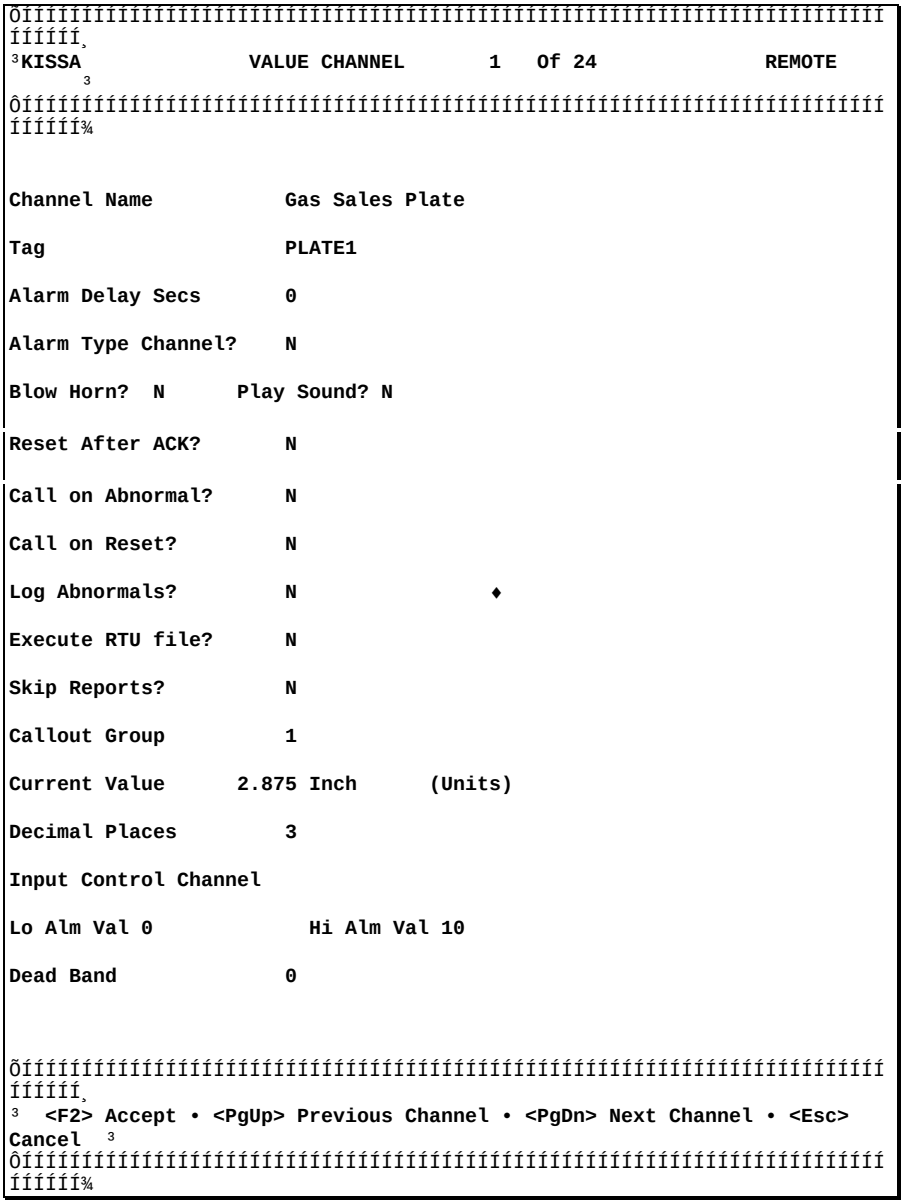
Units	Type of units this channel represents
Decimal Places	Display and transfer precision
Input Control Chan	Status input that controls this channel
Lo Alarm Value	Low setpoint in engineering units
Hi Alarm Value	High setpoint in engineering units
Deadband	Margin used before clearing alarms

The Decimal Places parameter determines how a channel will be displayed and transferred to another unit. This parameter is separate for each channel. If the parameter is 0 or greater, than the program will format the text version of the channel to the specified number of decimals. If the setting is negative, then the program will use a single, global decimal place count established with the SET PLACES x command. Therefore, most channels can be set easily by setting the system default and leaving each channel at the system default setting. Certain channels, such as orifice plates, will display better with a higher number of decimal places. Other channels, such as AGA-3 flow rates, will display better with 0 or 1 decimal places. These channels can be individually configured as desired, while the majority of the channels can be left at the default system setting.

Note that setting the decimal places will limit the default precision of the value on the current system as well as on any system that receives the value via a transmission. If only 1 decimal place is used at an RTU, then the transfer will only provide 1 decimal place as received by the HOST. Setting a higher number of places at the HOST will not result in any additional precision because the precision has been removed at the RTU prior to transmission.

The Input Control Channel parameter is optional and can be used to specify a channel which will control

this channel's ability to do certain automatic actions. This will prevent inadvertent alarming of value inputs (or any of its derivatives like analog or meter) when certain conditions prevent normal operation of the system being monitored. For example, it is undesirable to get a low oil pressure alarm from an engine that is not running. Assigning a control input that is tied to the engine run status will prevent the analog channel from operating when the engine is not. Leaving the Input Control Channel entry blank defeats this function.



Any type of channel can be used as an Input Control Channel. The control rule is that the input control channel must be in the normal state in order for the affected channel to operate. If the control channel is abnormal, the alarm mode functions of the channel are temporarily defeated.

The Deadband parameter provides a stage to the alarm process to accommodate points in alarm that are in the process of returning to normal. The deadband is the range that the value must exceed *in addition to*

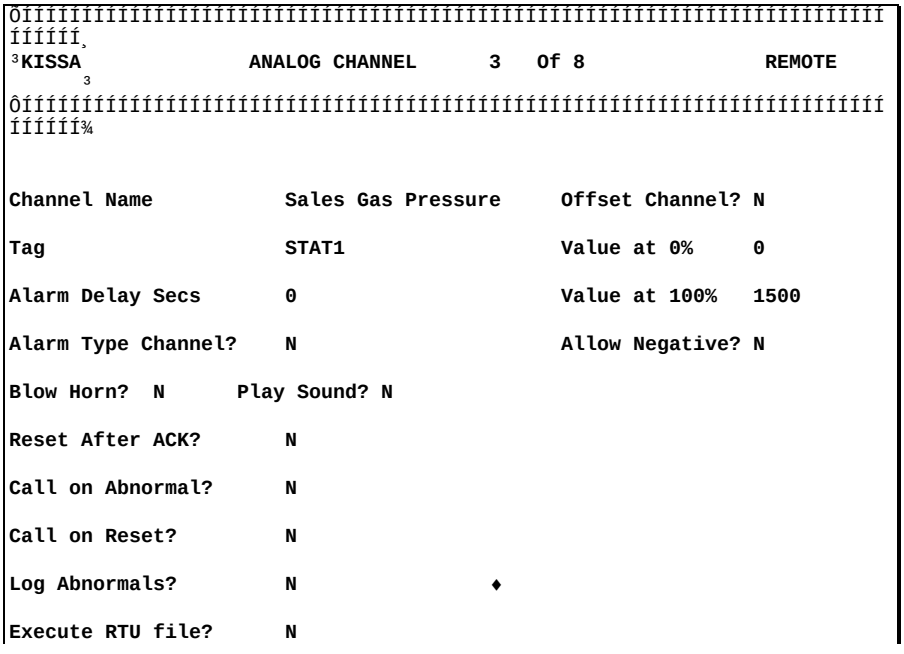
the normal setpoint before it is allowed to clear. This parameter is provided to reduce nuisance alarms on channel types that hover near the alarm point. Without the deadband, the channel would continually re-alarm in response to small changes in the channel value. The deadband allows for customization of each point to determine how much of a change is required to clear an existing alarm. If a channel has a deadband of 0 then this parameter is ignored. The deadband applies to both high and low alarm conditions.

Lets consider a battery voltage channel that has a low alarm setpoint of 11.5 volts. As the voltage drops past 11.5, the point will alarm as expected. If the deadband is 0, then the point will clear when the value rises back to 11.5 or higher. But, this may cause a re-alarm problem if the value stays in the vicinity of 11.5 volts. Each time it dips slightly below 11.5, a new alarm would occur, and each time it rises the alarm would clear. To prevent this, a deadband of 1 volt could be used to prevent the alarm from clearing until the voltage rose past 12.5 volts (11.5 setpoint plus the 1.0 volt deadband).

9.7 ANALOG INPUT PROGRAMMING

Analog inputs are derived from value channels to produce an engineering units value related to some form of real-world signal. These signals are usually associated with pressures, temperatures, voltages, and other process values. However, any signal that can be scaled from one point to another can be connected to the analog input system. Less common examples include Ph, specific gravity, moisture content, and valve position. The only requirement is that the input be linear between two fixed values (such as 0-1000 Psi or 50-150 Degrees).

Analog inputs enter the system through some form of analog to digital (A/D) converter. These hardware devices convert an electrical signal representing the field condition into a number that can be processed by the computer. The initial conversion from an electrical signal into a number is called the *raw conversion.* The raw analog input value is processed by the Driver Task into a signed 16 bit integer, providing an overall range of +-32K. This will accommodate A/D cards with up to 16 bit accuracy, although devices with 8 or 12 bits are more common. The hardware driver for the particular system will scale the actual input value up to a 16 bit signed integer. This 16 bit integer is the "raw" value of the channel. From this point on, the SCADA software treats all of the inputs the same regardless of their actual origin.



```
Skip Reports?          N

Callout Group          1

Current Value      428.5 Psig      (Units)

Decimal Places         1

Input Control Channel

Lo Alm Val 0             Hi Alm Val 100

Dead Band              0



ÕÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¸
³  <F2> Accept • <PgUp> Previous Channel • <PgDn> Next Channel • <Esc>
Cancel  ³
ÔÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¾
```

The program automatically handles straight linear (i.e. 0-10 volt) or offset (4-20ma) type channels. The difference is whether 0% input occurs at 0% of full scale (as in voltage readings) or at 20% of full scale (as in a 4-20ma transmitter). The program will take care of all of the calculations necessary to figure the actual input setpoints based on the user's input which is in engineering units.

Each channel can also block negative numbers from being generated during the conversion process. This is desirable for pressure and other inputs that may show slightly negative numbers when idle. This is the default setting, and will cause numbers that otherwise would be less than zero to appear as zero. If negative numbers are desired, set the Allow Negative Option to TRUE during channel configuration. This setting only affects the raw signal conversion, and a negative value can still be placed in a channel with a CALC or DATA statement.

Note that all of the parameters associated with analog inputs are at level 3 and only affect the raw calculation at the RTU where the conversion takes place.

Offset Channel	Is this a 0-x or an x-y type channel.
Value at 0%	Engr. value at 0% input
Value at 100%	Engr. value at 100% input.
Allow Negative	Are negative numbers allowed for this channel.

As an example of an analog setup, consider a 50-150 degree temperature transmitter that is connected to an RTU with a 4-20ma signal. This device is an offset input because the low value occurs at 4ma, not 0 ma. The 0% value is 50 degrees, and the 100% value is 150 degrees. We will not want negative numbers for this channel.

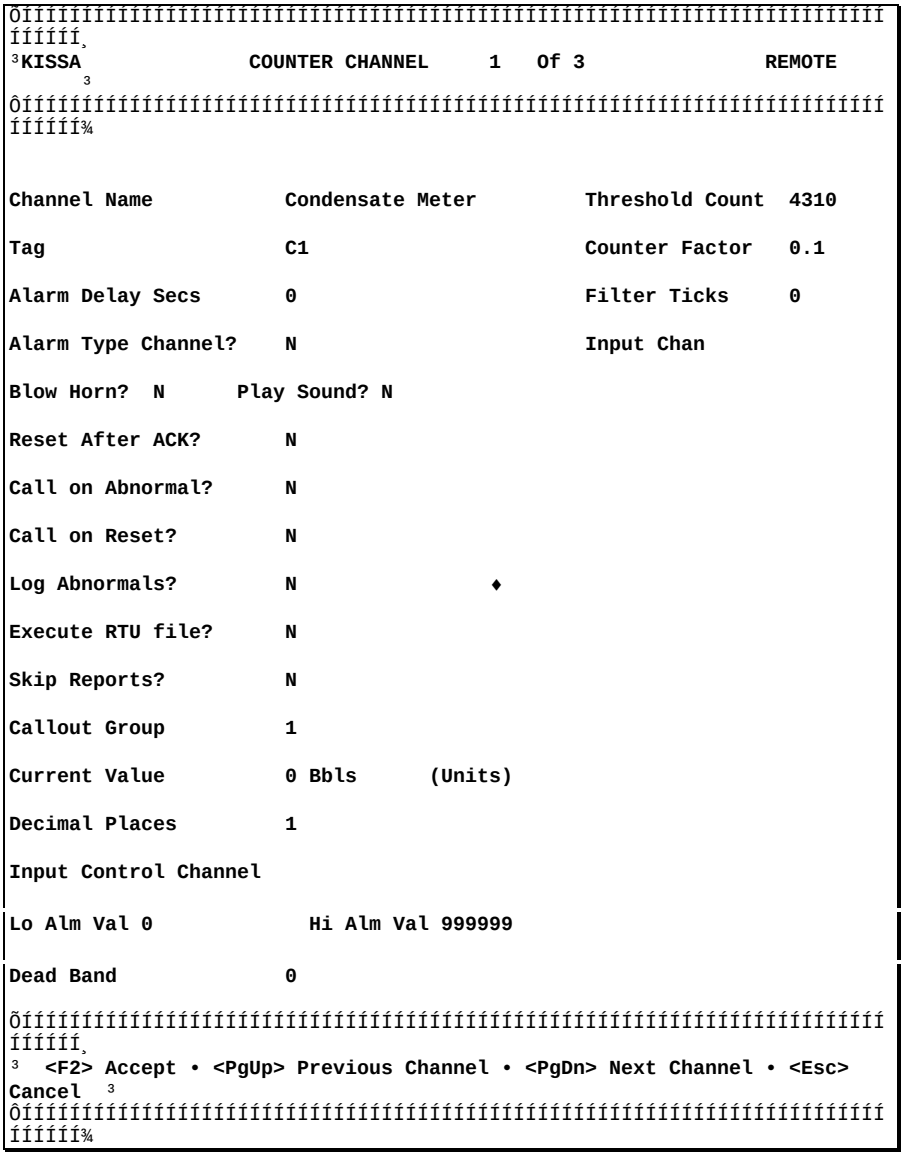
Another example is a battery input that ranges from 0 to 20 VDC. This may actually be a nominal 12VDC input that is scaled with resistors to produce the actual voltage needed by the analog input (perhaps 0-5VDC). The input would be programmed as Not offset, 0% of 0 VDC, 100% of 20VDC, and no negative allowed.

9.8 COUNTER INPUT PROGRAMMING

Counter inputs channels are designed to accumulate events of some sort beginning at a particular moment in time. The channel can accumulate high speed pulses entering a hardware counter circuit, or it can accumulate logical state changes for any other channel type. This allows a counter channel to totalize high

speed pulses with a counter input card, or to count slow speed pulses appearing at a status input or output channel. Regardless of the count source, the counter channel accumulates what is called an "event."

The value of the channel represents the number of events that have occurred since the channel was last reset. These events may be something simple like pump strokes, or it may be more complex like turbine meter pulses. We probably don't care about how many turbine blade pulses have occurred, but we would like to know how much flow caused the blades to move. The Counter factor value is used to convert any number of events (pulses) into another form of units, such as total flow.



The hardware counter input system is very flexible and can monitor very slow to very fast (8000 Hz) input signals. The flexibility of the rate depends on the hardware that is used. The signal level can be from millivolt turbine meter inputs through high level opto-isolated voltage inputs. The counting occurs in two steps. The first step accumulates the raw input pulses until a threshold is reached. At each threshold, one unit of

measure is added to the value of the channel.

For example, suppose we have a meter that provides 100 pulses per gallon. We can set up the channel to show whole gallons by entering a threshold of 100 as being equal to a value of 1. Or, we can set it up with a threshold of 10 to equal a value of 0.1. This provides a resolution of 1/10 gallon rather than one gallon as in the previous setup. More complicated setups provide for converting gallons to barrels or pounds to cubic feet. The basic concept is the same. How many ticks equal one unit? And how much is one unit? Those two values must be known before the channel can be programmed.

The counts from the actual hardware counter circuit are processed on each system tick, 1/18 of a second. The count is added to a software register that is 16 bits wide, allowing 32K counts to be accommodated. The threshold value is specified for each counter channel that provides the number of whole counts in the software register that will equal one unit of measure. The units of measure are accumulated at the value level by adding one unit to the existing total each time the threshold is exceeded.

Ok, lets look at another example. Consider a turbine meter that provides 5000 pulses per gallon of fluid. The fluid weighs 5.2 pounds per gallon. We want the counter to indicate total flow in pounds. We set up the threshold value as 5000, because this is the number of counts that represent one unit of measure based on the design of the meter. The conversion factor will be 5.2, so that the resulting value will be in pounds and not gallons. We count 5000 ticks to get one unit, and each unit adds 5.2 to the total. We could also set it up so that 500 ticks is one unit, and each unit is 0.52.

One additional parameter called Filter Ticks is used in processing noisy, slow pulsing channels. This parameter, if non zero, sets the system to de-bounce counter inputs and add a delay between allowable inputs. This effectively reduces all counter inputs received in one burst to a single pulse, and locks out any further pulses until the specified number of system ticks has passed.

For example, a system with a parameter of 18 that receives 17 random pulses when a relay closure occurs will count all 17 as a single pulse and will wait approximately 1 second (18 ticks) before permitting another pulse to be counted. This is very useful for debouncing real world inputs such as relay and switch contacts as well as noisy solenoid circuits that are monitored as pulse inputs.

All counter parameters are at level 3. The special parameters for counter inputs are:

Threshold Count	Number of counts to = 1 unit of measure
Counter Factor	Engineering units conversion
Filter Ticks	Ticks to wait between counting pulses
Input Channel	Optional SCADA channel to be monitored for 0-1 transition.

9.9 FUNCTION CHANNEL PROGRAMMING

The Function channel is a virtual channel that is not directly associated with any real input point. They exist only in the "mind" of the computer. The intent is to take another channel value and provide a special function to it so that another value is produced. This new value is related to the original channel in a specific way. The current functions that are available are Rate (frequency), Average, and Difference. A procedure option can also be used in addition to any of these types.

1. The Rate function will monitor an input in order to determine how much the input changes over a fixed period of time.

2. The Average function is similar except that it calculates the average value during the sample period.

3. The Difference type simply calculates the difference in the input value from one pass to the next. This is useful in detecting rate of change values for the specified input channel.

4. The Procedure Setting makes the Function act much like an automatic Timer channel that calls a TSP

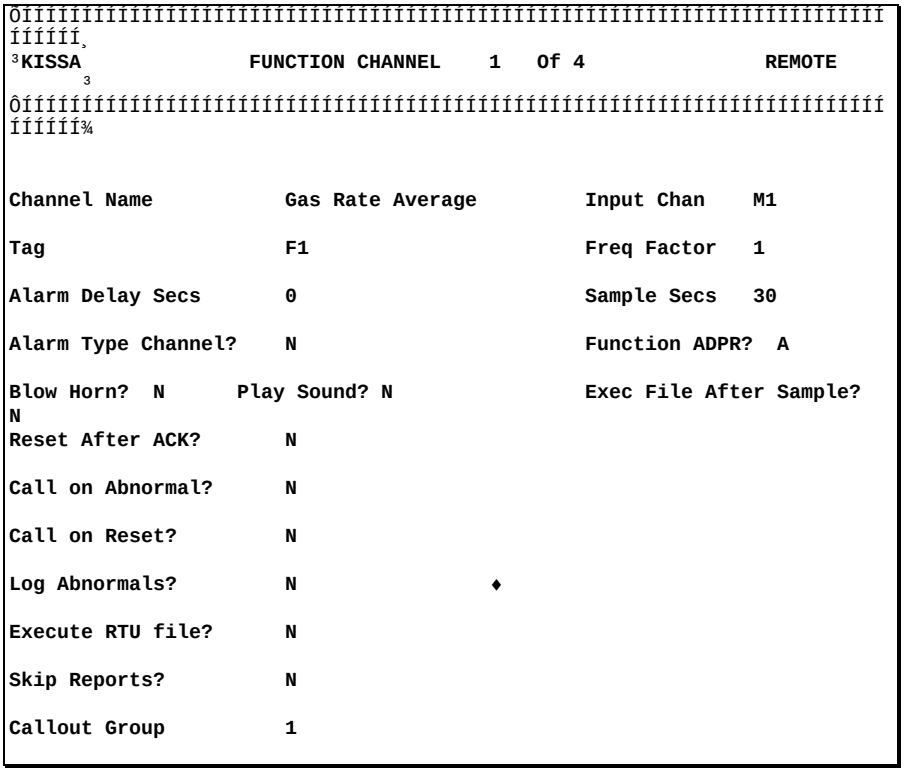
procedure once every cycle. The Procedure name is the same as the channel tag name and is processed by the Utility task whenever the time period of the channel expires. But unlike the normal Timer channel output, the Procedure function passes additional information on the command line that is sent to the Utility task for processing. The parameters are:

0 Tag Name (same as procedure name)
1 Mode number of 3 (0=reset, 1=alarm, 2=reserved)
2 RTU name
3 Current value of function channel
4 Current value of input channel
5 Tag name for input channel
6 Function factor
7 Seconds for Function update period

Any channel type can be used as an input source for the function channel, although the Rate method is most likely to be used only with a counter input channel.

The Rate process operates by reading the input value at the beginning and end of the sample period and then using the difference to calculate the frequency. The Average calculation occurs each second to produce an average over the sample period. So, the Rate method only uses the beginning and ending values while the average method uses the value at each second of the sample period.

Some inputs occur randomly, and have no real rate associated with them. Others, like continuous flow turbine meters, have a reasonable rate that can be calculated. The frequency calculation basically examines the number of pulses per time period. The time period for each sample can be set to any number of seconds, with 0 disabling the rate calculation. The length of the sample period is selected to give a reasonable time for a typical number of pulses to occur. Very fast inputs can use each second, while somewhat random pulses may use several minutes worth of seconds to allow for variations in the rate to be smoothed out.



```
Current Value        8.2 MMCF/Day  (Units)

Decimal Places         1

Input Control Channel

Lo Alm Val -1            Hi Alm Val 999999

Dead Band              0



õíííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííí
ííííííí¸
³  <F2> Accept • <PgUp> Previous Channel • <PgDn> Next Channel • <Esc>
Cancel  ³
ôíííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííííí
ííííííí¼
```

Each Frequency channel has a factor that will be multiplied by the number of counts/second accumulated during the most recent sample period. This factor can do engineering unit conversion as required to obtain the results in any engineering units. The factor may also account for time period changes.

Consider the turbine meter mentioned in the Counter programming section above. If we examine the counter over a 10 second period, the program will determine the difference in the count that occurs in 10 seconds. The count difference will be divided by the number of seconds in the period to get a value related to the number of gallons of fluid that went by during that time. If we want to get a flow rate based on Barrels per day, an additional conversion factor is needed to do the necessary juggling of the numbers.

This factor will account for the fact that there are 86400 seconds in a day, and that there are 55 gallons per barrel. The factor will be calculated as follows:

```
            Gallon           Secs     1   Bbl
FACTOR = X ÄÄÄÄÄÄÄÄ * 86400 ÄÄÄÄÄ * ÄÄÄ ÄÄÄÄÄÄÄÄ = 1570.9
              Sec             Day     55  Gallon
```

The factor of 1570.9 will provide the rate calculation in bbl/day, and this unit of measure can be specified for the function channel.

Special parameters for function channels are:

Input Chan	Channel where the input comes from
Freq Factor	Volume and time conversion factor.
Sample Seconds	Sample period given in seconds
Function ADPR	Function Type: Average, Difference, Rate, or Procedure

If a channel is set up as an average type channel the rate will be based on the average value over the sample period. Otherwise, the rate will be based on the difference in value over the sample period.

9.10 TIMER CHANNEL PROGRAMMING

Timer channels are not directly associated with any physical devices but exist entirely in software. When a timer is started, it loads the time value into a software counter. **This counter is decremented (or incremented) once per second for starting times over 100 seconds, and every 50ms for starting times less than 100 seconds**. Timers can be operated up or down, single-shot or one-shot. Automatic control of other channels (usually physical outputs) is also easily accomplished with a separate setting for each timer channel.

The special parameters used for timer channels include:

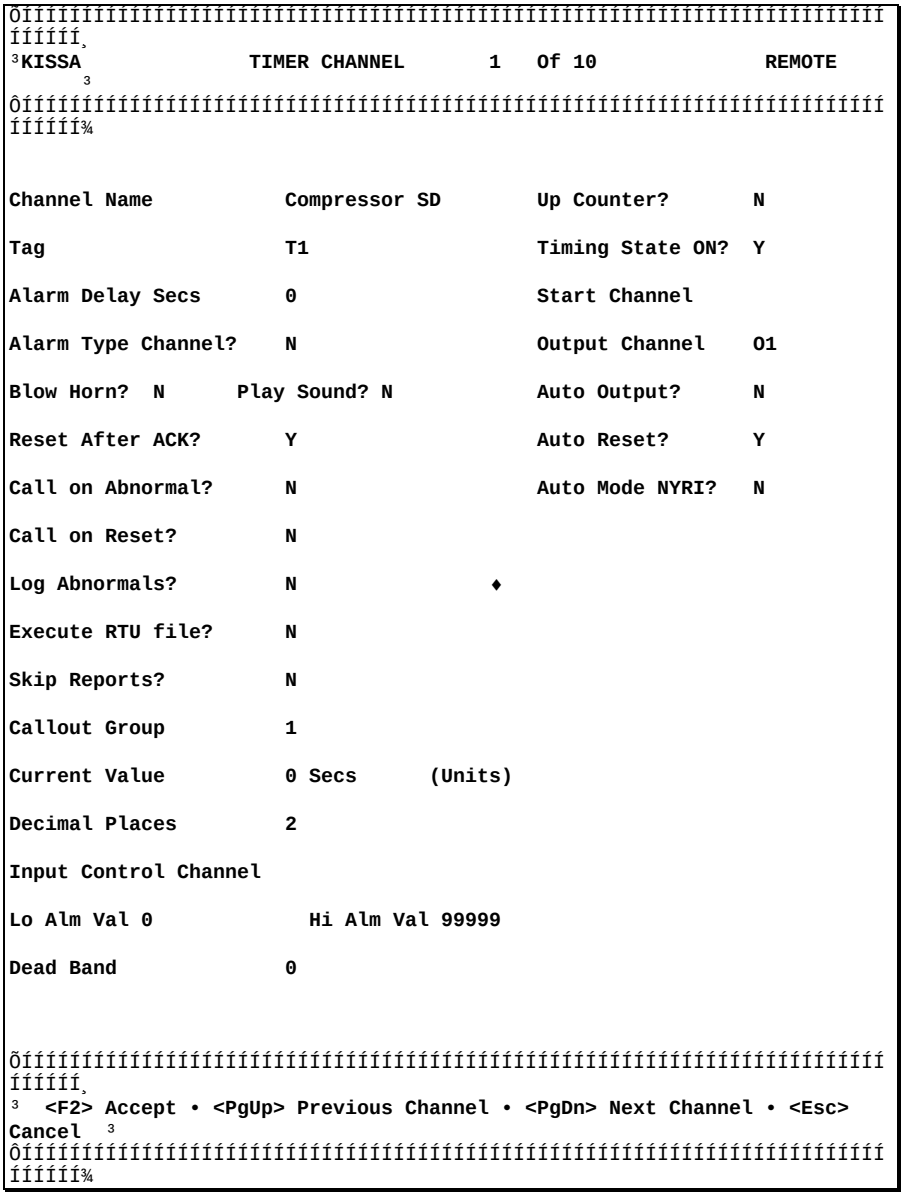
Up Counter	Does timer count up or down.
Timing State On	Is output on (not off) during timing period.
Start Channel	Channel which start timer.
Output Channel	Channel to be set to 1 (usually an Output)
Auto Output	Does output follow control input.
Auto Reset	Is count restarted each time.
Auto Mode NYRI	Auto restart mode of No, Yes, Reset, or Interval

The Up Counter parameter determines whether the timer is incremented or decremented when it is timing. A timer channel that is set to count downward will stop timing if its value reaches 0. A timer that counts upward will never stop timing simply by reaching a certain value. If an upward counting timer is left timing indefinitely it will eventually exceed the counter's limits and take on an unpredictable value. The timer's maximum limit is so large that if this ever happened, there is probably an error somewhere else that caused the timer to start counting at the wrong value.

During the timing period, the timer can be restarted (with CALC TIMER = ON) which will reload the time value in the high setpoint. This is useful for fail-safe applications like hurricane evacuation timers.

If a timer is set up as an alarm type channel the conditions for an alarm depend on whether the timer is counting up or down. A timer that counts downward will go into alarm only when its value becomes equal to or less than the value of the low alarm setpoint. A timer that counts upward will go into alarm only when its value becomes equal to or greater than the high alarm setpoint. After going into alarm, a timer channel will continue timing in its usual manner.

The Output Channel parameter can be used to manipulate any other channel during the timing period. If an output channel is specified, it will be set to 0 or 1 while a timer is counting. The state of the output channel is controlled by the Timing State parameter.



The Start Channel parameter allows another channel to control the starting of a timer rather than having to load it or use the CALC command. This option can have a timer start automatically (with its high setpoint value) when a status input or output channel is in its abnormal state, or when a value type channel is within its high and low alarm setpoints. This provides some interesting options when the output of one timer starts another timer. For example, a pulse output can be programmed by having the first timer turn on an output and trigger the second timer, which then turns the output back off when its time expires. This off-on cycling can also be accomplished with the Interval timer option explained below.

When an input channel starts a downward counting timer, the timer's count begins with the value of the high alarm setpoint. If the timer counts downward and stops when it reaches 0, it will automatically start counting again if the status of its start channel is still in the proper condition.

If the channel is an up counter, it starts automatic timing with the value in the low setpoint. Alarm actions will occur when the high setpoint is reached. However, unlike a down counter, an up counter will not

automatically reset when it reaches the setpoint but will continue counting up until reset by some external means. If the auto mode of the channel is YES, then it will restart automatically when a timeout occurs.

Timers set up for repeat interval will automatically cycle the output off and on. The on time will be equal to the high setpoint, and the off time is determined by the low setpoint. This function can be used to drive an output off and on according to the current values of the setpoints. These setpoints can be easily changed at runtime, simplifying manipulation of the timing periods.

Extra caution should be taken when specifying a Start Input channel that is a value type channel and not set up as an alarm type channel. The user will have no way of knowing if the value of such a channel has exceeded an alarm setpoint and then returned to normal. Therefore, it will appear to the user that this channel should start the timer again automatically when it reaches 0. Unfortunately, this will not work. The channel must be reset and acknowledged before it can start the timer again. This problem can be overcome by configuring the input channel so that is will automatically reset itself when its value returns to an allowable range.

The Auto Reset parameter is used to automatically set a timer's value to 0 if the optional Input Control Channel de-activates. This allows a timer to monitor a status input and only time-out if the status input remains active for a continuous period. Intermittent timing periods will not generate a time-out. If the auto reset option is not specified, the timer will time out after timing for any number of timing periods that total up to the timing period.

The Auto Output option determines how the output behaves in response to the Input Control Channel during the timing period. If false, then the output will remain active even during periods when the control input is inactive (and the count is holding). If true, then the output will only be active when the timer is actually counting down. When the timer stops, the output will go to the non-timing state. When the count resumes, the output will return to the timing state.

The Auto Restart option is used to control whether or not a timer channel is automatically restarted when its value reaches an alarm setpoint. Using this option will cause a downward counting timer to reset itself to its high alarm value whenever it reaches its low alarm value. The timer will automatically continue counting with a "full load.". Likewise, an upward counting timer will automatically reset itself to its low alarm value and continue timing whenever it reaches its high alarm value.

The Auto Mode NYRI options have the following meanings:

N	No - Do not do automatic cycling
Y	Yes - Do automatic cycling to restart on timeout
R	Recycle - Same as Yes (for compatibility with older versions)
I	Interval Timing with high and low setpoint values as on and off times.

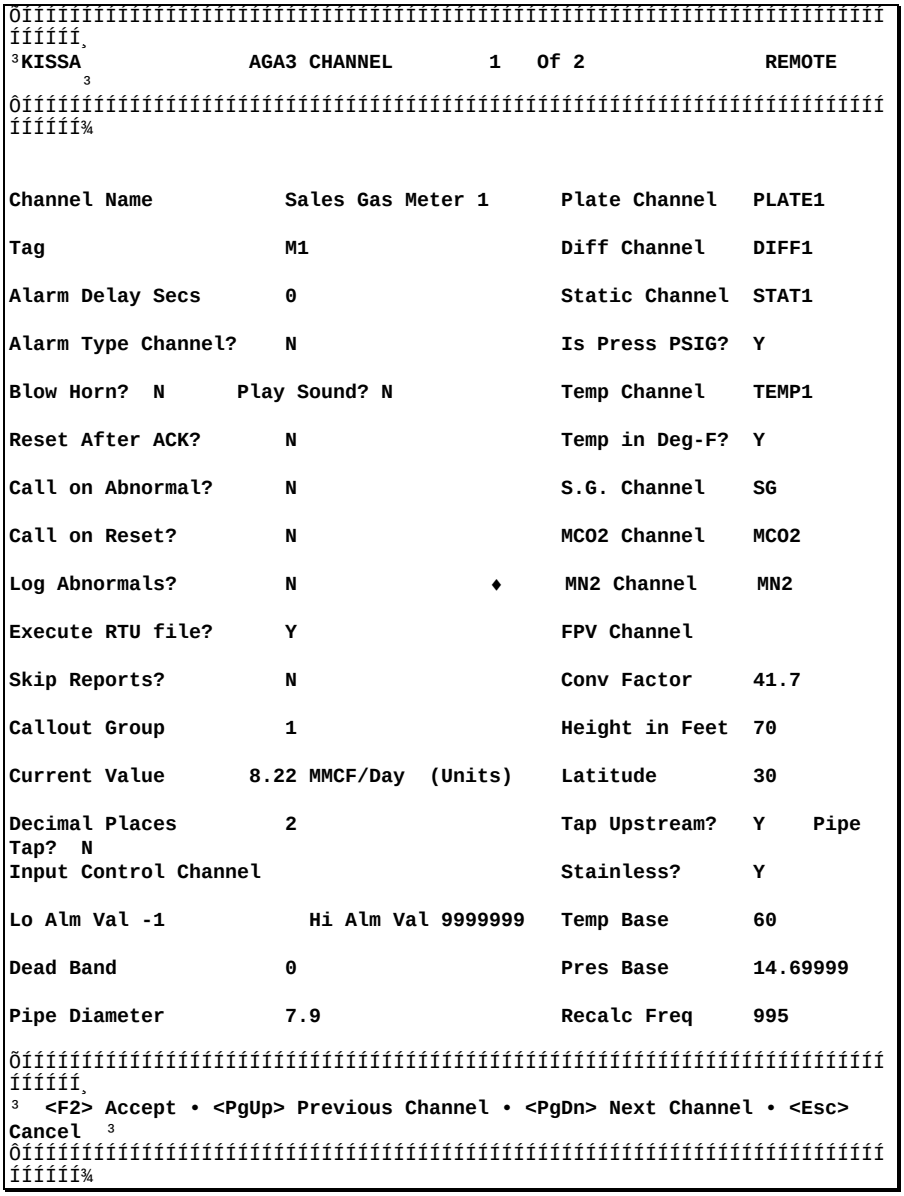
9.11 AGA3 METER PROGRAMMING AND ERROR CODES

The AGA3 meters are the most complicated type of channel in the system. This is due to the large number of parameters necessary to completely do the AGA3 calculation as specified in the AGA3 publication. Many of the parameters can be left at their default values which represent typical oil field gas production values. Only the values that are known should be changed. Of course, as more parameters are provided the validity of the calculation will improve.

The most important parameters are the physical installation specifications including pipe and plate sizes, tap location, and plate material. Also, the gas specific gravity must be fairly accurate in order to get reasonably accurate results. Other values, like percent CO2, are less important and can be ignored if desired.

The parameters are obtained at calculation time from two different sources. The parameters that rarely change (like the pipe diameter) are contained in the setup data for the particular channel. Other values that do change (like the plate size) are obtained from a designated input channel. This allows for on-the-fly modification of the AGA-3 parameters through changes to the associated analog, value, or other input channel.

The three basic real-time inputs that are used in the meter calculation are the static pressure, differential pressure, and temperature. These inputs may be assigned to any channel type, although analog and value channels are the most common. If a constant must be used (as when there is no actual temperature input), then a value channel must be set up that contains the desired constant.



Two other channel- related inputs are the plate size and the specific gravity. Valid channels must be specified for these inputs. If invalid inputs are specified, the software will not proceed with the AGA3 calculation and will instead generate an AGA3 error value.

Two other optional channel-related inputs are the mole percent N2, and the percent CO2. The channel

designations for these values can be skipped if desired by leaving the field blank. If the channel for these two inputs is invalid, a value of 0 is assumed when the AGA3 calculation is made.

The final channel related input is the optional FPV factor which is the supercompressibility factor used in AGA3 calculations. Normally, this parameter is not needed and the program will automatically calculate the FPV factor from data provided by the other parameters (using NX-19 rules). However, if a valid channel is specified for this input, the program will skip the FPV calculation and simply use the value of the specified channel as the FPV factor. This is necessary for special gases that do not conform to NX-19 guidelines. For these gases, the Fpv will have to be obtained from some other source and supplied to the AGA-3 calculation via a special channel.

The special parameters for the AGA3 meters are:

Factor	Converts calculation from MCF/Hr. to desired time and engr. units. Typical value is 41.667 for (1000 / 24) MMCF/Day.
Channels	Input channels for Temp, DP, PSIA, SG, Plate Size, %CO2, %N2, and FPV.
Pipe Diam	Inside Diameter of pipe in inches.
Tap Location	Up or down stream taps.
Tap Type	Pipe tap or Flange
Plate Matrl	Stainless or Monel Plate material
Elevation	Height above sea level in feet.
Latitude	Degrees latitude of meter (getting picky!)
Temp Base	Temp at which gas value is equated (60-F).
Press Base	Pressure at which gas value is equated (14.7)
Calc Freq	How often complete calculation is performed.
Temp in F	True if temperature input is Deg-F, not Deg-C
Press in Psig	True if static input is in PSIG, not PSIA

The AGA3 meters can be calculated in a number of ways to accommodate various operating conditions. Adjustments are provided for remote and local calculation, and for the frequency of the complete AGA3 calculation. Each will be considered separately.

When a meter is calculated, a number of parameters are required to determine the final flow rate. If the values are accessed in the local system, then the meter value is calculated locally. This requires that all parameters be correct in the local system. It may be desirable to only maintain the values in the actual remote system, and simply download the calculated meter value from the RTU rather than calculate it locally at the Host. Either method is available.

If a meter point is specified as a remote point (by virtue of its belonging to a remote RTU), then the meter value will never be automatically calculated locally. Calculation locally will require the use of the AGA3 command either from the keyboard, or in an RTU command file executed by some task.

A Host computer can get AGA3 values in one of two ways. The most common is to download the calculated meter value in engineering units from a smart RTU directly into the meter channel at the Host. This lets all of the details of the calculation be handled by the RTU itself and reduces the Host to a simple display and alarm function. Upon receiving the values, the Host will scan them and generate alarms as required. The actual calculation will never take place at the Host. The Host will simply process the final answer downloaded from the RTU.

Another method is to download the input values from the RTU and perform the AGA3 calculation at the Host computer. This will require the use of an AGA3 command if the meter is set up as a remote point at the Host. If the point is configured as local, then it will be calculated constantly just as if it were at an RTU. When doing the calculation at the Host, it is necessary to have all of the meter data correctly installed at the Host. This is not necessary for calculations done at the RTU. *NOTE - The calculate at the Host method for getting AGA3 values is required when using a Type 2000 "dumb" RTU. This is because the Type 2000 RTU cannot perform AGA3 calculations at the RTU itself.*

If a meter channel is marked as a local (non remote) point, then its value is calculated by the SCAN

task that normally runs continuously. This would be similar to local analog channels. However, because of the complexity of the AGA3 calculation, an adjustment is provided to control how often the complete calculation is done. A partial calculation, which takes into account only pressure, differential, and temperature, can be done much quicker than the complete calculation. So, depending on the nature of the meter being installed, it may be possible to defer the complete calculation until a lesser parameter is changed.

The frequency of calculation is important to avoid overloading low power processors with more than a few meters. For example, a PC level machine with 5 meters running continuously would suffer in performance because the CPU would be so busy doing AGA3 calculations. However, 286 and 368 machines operate more quickly and are not affected nearly as much by the AGA3 calculations.

These lesser parameters are any ones other than the 3 primary inputs. A meter whose SG, plate size, Pct N2, etc. change infrequently can use the quick update always. However, a meter with an on-line gas analyzer providing real-time gas quality inputs will have to do the complete calculation on each pass, or as often as desired.

The calculation frequency is based on the number of passes by the SCAN task, which is difficult to predict due to system dynamics. A setting of 0 means that the full calculation is never automatically done, and must be initiated by programming the channel, enabling it, or using the AGA3 command. A setting of 1 means that the full calculation happens on each pass. A number higher than 1 means that the set number of cycles will occur with partial calculations before the complete calculation is done and the counter is reset.

Any command files that cause changes in any parameters (like the plate size) should issue an AGA3 command so that the complete calculation will occur right away. Also, daily command files can include an AGA3 command to insure that the complete calculation happens at least once per day.

The conversion factor included in the setup is used to convert the standard AGA-3 calculation, which is in CF/Hr, to the desired flow units such as MCF/Day. The factor is divided into the AGA-3 calculation, so a conversion to MCF/Day would require a factor of 1000/24, or 41.667. The system defaults to a factor value of 1000 and units of MCF/Hr to match the AGA-3 document requirements.

The AGA3 calculator will detect many errors and halt the calculation where the error occurs. A temporary internal variable is used to track the error during each calculation attempt. The error result from the last attempt is held as the externally available error code throughout the next attempt. This avoids the possibility of downloading an incorrect (or missing) error code if the SCAN command happens to catch the calculation in progress at a point where the code was set to 0. In the case of an error, the current rate is shown as 0 and the error code is shown in the AGA3 meter display. A scan on a meter channel that is in error will cause the error number to be downloaded as a negative number.

The following is a list of error codes for AGA3 calculations. The terms used refer to the terms used in the actual API publication describing the calculation. These codes are also contained in a small text file called AGA3.HLP. This file can be viewed online using the EDIT or TYPE command or by entering HELP AGA3.

AGA-3 ERROR CODE SUMMARY

1	-	Pipe diameter <= 0
2	-	Base pressure <= 0
3	-	Local gravity factor used in calculating the Location Factor (Fl) is <0
101	-	error reading orifice plate channel
102	-	error reading specific gravity channel
103	-	orifice plate size <= 0
104	-	orifice is greater than the pipe diameter
105	-	specific gravity is <= 0
201	-	error reading differential pressure channel
202	-	error reading static pressure channel
203	-	error reading temperature channel
204	-	differential pressure < 0

205 - psia calculated to be < 0
206 - Reynolds Number Factor (Fr) = 0
207 - Temperature used in calculating the Flowing Temperature Factor (Ftf) is = 0 when converted to Rankine.
208 - error calculating parameter E (which is used to calculate the Supercompressibility Factor (Fpv)) with the range of adjusted pressure and adjusted temperature being 85 <= temp <= 240 and 0 >= press <= 2000
209 - error calculating parameter E (which is used to calculate the Supercompressibility Factor (Fpv)) with the range of adjusted pressure and adjusted temperature being -20 <= temp < 85 and 1300 <= press <= 2000
210 - error calculating a factor used to calculate Fpv.
211 - calculated gas flow < 0 although no other errors occurred.

9.12 TOTALIZER CHANNEL PROGRAMMING

Totalizer channels receive input from another channel and accumulate the total amount by calculating rate times clock time. Inputs to the totalizer can be any type of channel, but will most likely be an Analog or Meter channel. Totalizers have the same alarming and control functions as other channels, and programming is very similar.

Totalizers start at 0 at a time and date that is automatically tracked by the channel. The total will continue to accumulate until reset by some source, normally a menu selection or a time of day event. For example, the Agenda system may be used to log and clear the Totalizers each morning after transferring the previous day's totals to value channels for later reference. When the Totalizers are cleared, they will automatically set their time and date so that they accurately reflect the totalized flow from a known point in time.

The complete formula used in calculating the Totalizer value each second is as follows:

$$\text{Quant Units} = \frac{\text{Rate Units}}{\text{Rate}} \times \frac{\text{Quant Units}}{\text{Rate Time Period}} \times \frac{\text{Seconds}}{\text{Rate Units}} \times F1 \times \frac{\text{Rate Time Period}}{\text{Quant Units}} \times F2$$

```
ÕÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¸
³KISSA               TOTALIZER CHANNEL   1   Of 2               REMOTE
      ³
ÔÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¾


Channel Name           Sales Meter            Input Channel  M1

Tag                    Q1                     Min Value      0.1

Alarm Delay Secs       0                      Factor         86400

Alarm Type Channel?    N

Blow Horn?  N      Play Sound? N

Reset After ACK?       N

Call on Abnormal?      N

Call on Reset?         N

Log Abnormals?         N                ♦

Execute RTU file?      N

Skip Reports?          N

Callout Group          1

Current Value      4.827 MMCF      (Units)

Decimal Places         4

Input Control Channel

Lo Alm Val 0             Hi Alm Val 100

Dead Band              0


ÕÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¸
³  <F2> Accept • <PgUp> Previous Channel • <PgDn> Next Channel • <Esc>
Cancel  ³
ÔÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¾
```

The factor that needs to be entered is F1 * F2. This takes care of both engineering units conversion as well as time units to produce a number that is just quantity, without a time element. The example below will make this somewhat clearer.

The program totalizes every second by taking the current rate value (from whatever source), dividing by the factor, and adding the result to the previous total. The incremental amount should be the quantity accumulated during the previous second, assuming that the rate was constant during the second. Okay, okay. Look at an example and see if it becomes a little clearer.

EXAMPLE: A gas meter is programmed to show a rate as millions of cubic feet per day (MCF/Day). We want the totalizer to show thousands per day (MCF/Day) because the flow is fairly low and we want to see the numbers change more often. Because the totalizer runs each second, the factor is calculated as:

```
                    Secs                     MMCF
   F1 = 86400 ÄÄÄÄÄÄ      F2 = 0.001 ÄÄÄÄÄÄ
```

```
                    Day                    MCF

          Factor = F1 * F2 = 86.4
```

Using the factor of 86.4 will produce a totalized value in MCF from a rate of MMCF/Day. If the current rate is 2.15 MMCF/Day then a one second increment is 2.15/86.4= 0.0249 MCF. This is the amount added to the total for the previous second.

EXAMPLE: For a more complicated value, consider a meter that is set to show Gallons/min. We want to totalize in pounds, and the fluid weighs 6.5 pounds per gallon. The factors are:

```
                 Secs             1   Gallons
      F1 = 60 ÄÄÄÄÄÄ    F2 = ÄÄÄÄÄ ÄÄÄÄÄÄÄÄ             Factor = F1 * F2 =
9.23
                  Min           6.5  Pounds
```

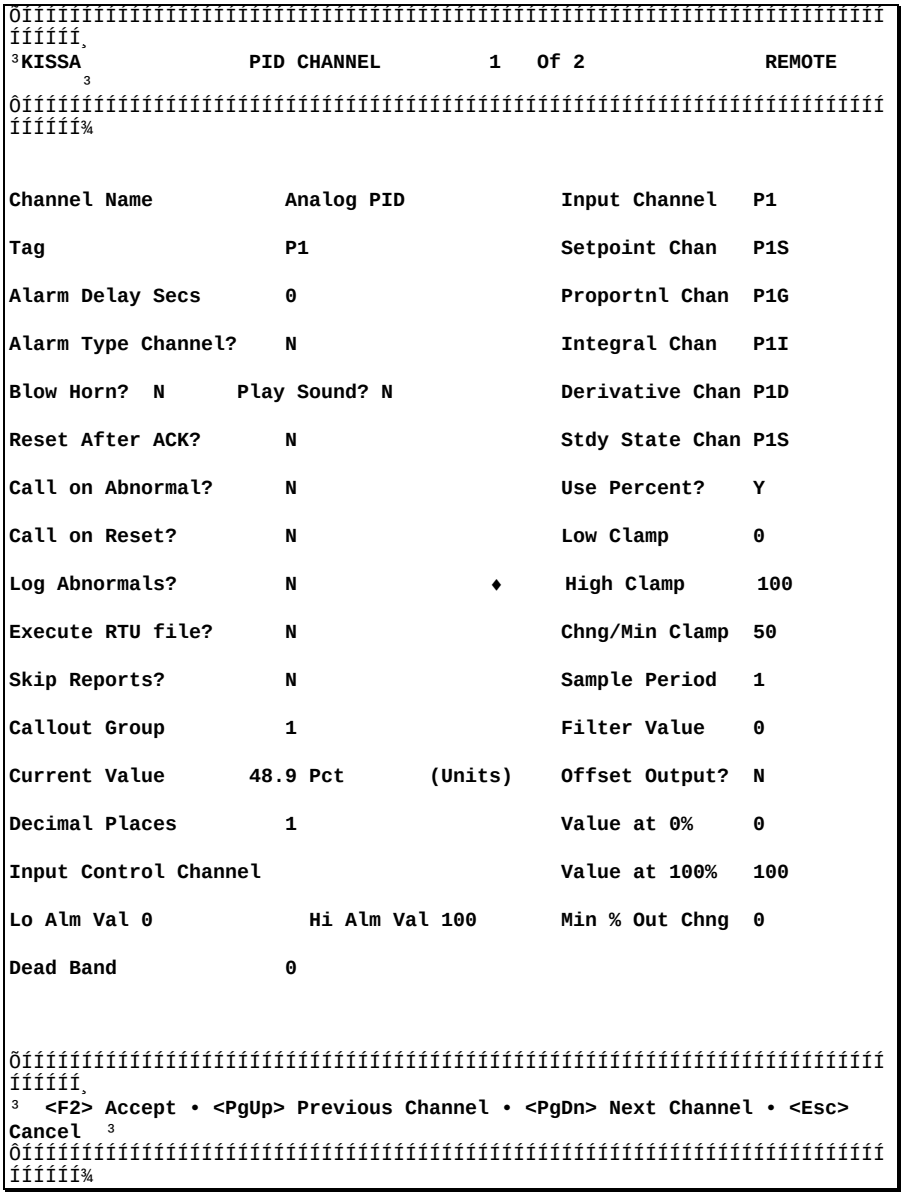
Using the factor of 9.23 will produce a totalized value in pounds from a rate of gallons/min.

The special parameters for totalizer channels are:

Input Chan	Channel where totalizer gets its input
Min Value	Minimum value to be considered a valid input
Factor	Used to convert input into desired rate

9.13 PID CHANNEL

The PID channel type is described in detail in section 19 of this document. Configuration of the PID channel type is similar to other value type channels.
The configuration screen is as follows:



9.14 CHANNEL REMOTE, HOLD, AND ENABLE SETTINGS

Each channel has 3 temporary settings that control its local calculation and alarm processing abilities. These are the Remote setting, the Hold setting, and the Enable setting. All of these settings can be altered with the CHANGE command and the Hold setting can be altered with the HOLD and UNHOLD commands. The alarm capability of each channel can be changed with the ENABLE and DISABLE commands.

The REMOTE setting controls automatic raw to engineering units conversion for each channel type. Local channels are converted by each pass of the SCAN task, while remote channels are assumed to be downloaded only. Remote channels are only checked for alarms when new data is detected, normally just after a download.

Each channel can represent data that is either locally accessed or is downloaded from another unit. There is very little difference in processing either type, although some controls are provided to determine low level conversions and alarm processing. Generally speaking, channels not directly connected to a system will not be locally calculated.

For example, analog inputs are generally downloaded from an RTU to a HOST, with the raw A/D conversion being done at the RTU. Although it is possible to have the RTU download the raw binary data and then do the conversion at the HOST, this is not necessary. AGA3 meters, however, are more complicated and may require processing at the HOST as well as the RTU. In this case, an AGA3 channel can either be changed to a local channel or it can be calculated at download time with an AGA3 command.

The HOLD setting determines if any raw to engineering conversions are placed in the current value for a channel. Placing a channel on HOLD temporarily stops the setting of the current value, as is often required during calibration or end device failures. For example, an AGA3 meter can be placed on hold while its input channels are calibrated. Any totalizer using the meter value as an input would continue to see the same number all during the calibration process. No channels are on hold by default.

The ENABLE setting determines a channel's ability to respond to the alarm mode settings. If a channel is enabled, it can blow the horn, cause callouts, etc. If disabled, then the channel can go into alarm or whatever without causing any automatic alarm mode actions. Channels are disabled during calibration or during end element failures to prevent unnecessary alarm activity. All channels are enabled at load time.

These three settings are saved to disk for each channel whenever a data image is saved. This allows the settings for all channels to be recovered if the program is ever stopped and restarted. See the section called AUTOMATIC VALUE SAVES for information on the data image.

Here's some examples:

```
HOLD M1           ; Freeze Meter 1 during calibration
CHANGE A1:a5 R-   ; Make these points local (not remote)
UNHOLD M1:m5           ; activate all meter channels
CHANGE S1:s5 E-   ; disable channels during maintenance
CHANGE a1:a3 E+   ; re-enable these points
ENABLE      A1         ; enable battery voltage alarm
DISABLE A11:a20   ; turn off alarms for all compressor points
```

10 COMMUNICATIONS AND DISPLAY TERMINALS

10.1 SERIAL COMMUNICATIONS

A serial port is a standard computer I/O device that takes data in 8 bit (one byte or character) form and transmits it one bit at a time to a similar device on the receiving end. The receiver reassembles the byte into an 8 bit unit and transfers it to the receiving computer. This allows two computers to communicate with 8 bit data over a single communications link. The tradeoff is the increased time required to transfer the information one bit at a time rather than one byte at a time.

All SCADA computers have serial ports. The RTU/SCADA system uses serial ports for many types of remote communications. The serial port of the PC usually connects to a modem or other data transmission device so that is can send information to a distant unit. The unit on the other end of the link has a similar device that receives the information and feeds it into the computer.

SCADAWARE also has the ability to operate via the PC's local console, or via a remote CRT terminal (or a remote PC emulating a terminal). The communications is normally done via one of the PC's serial communications channels, commonly referred to in the PC world as COM1 (pronounced "COM ONE") through COM4. This section is a very basic introduction to serial communications. Numerous texts are available on the subject for those wanting a more in-depth look at the world of communications.

The rate of bit transfer is called the "baud" rate, expressed in bits per second. Therefore, 1200 baud refers to a maximum rate of 1200 bits per second. The serial communications method used by the PC is called Asynchronous, meaning that characters can be transmitted at any time with an indefinite time period between characters. The bit spacing within characters must be exactly 1200 bps or else data errors will occur. To detect the errors and synchronize the transmission, additional bits are added to the actual character called Start and Stop bits.

These bits surround the actual character being transmitted, so it ends up taking 10 bits (1 start + 8 data + 1 stop) to send one 8 bit byte or character. So, the maximum character transfer rate (at 1200 baud) is 1200 / 10 = 120 characters per second. The device that does this is called a Universal Asynchronous Receiver Transmitter, or UART (YOU-ART). The particular one used in personal computers is the National Semiconductor NC8250 or some variant thereof. The PC system supports baud rates from 300 - 9600.

10.2 SERIAL PORT STANDARDS (RS-232)

An industry standard called RS-232-C is the common method of connecting serial devices to PC's. It was originally intended for communications between teletype devices over fairly short distances (25') and at fairly slow speeds (9600 baud max). Later versions, however, have been developed to overcome these limitations. For the purpose of the RTU/SCADA system, only the normal RS-232 type connections will be considered.

The voltages used in the RS-232 standard leave a lot of room for voltage drops and other connection problems. To do this, the standard uses a nominal 12 volt signal, both negative and positive, to represent a false (0) and true (1) signal level.

There is no simple explanation of which voltage is true and false, because different pins react accordingly. For example, when no data is being transmitted the data lines (2 and 3) are at -12 volts. When no carrier is being detected, the carrier detect is at -12 volts. However, the data lines are sending a true (1 or "Mark") when idle, meaning that -12 is a 1 to the data line. The carrier detect is sending a false using the same voltage. This is the result of the negative logic used on the control signals, and really adds to the confusion. When in doubt, draw a diagram to help make things clear.

In actual practice, most devices are not very picky about the signal levels. Most will work with anything up to 30 volts as long as the positive and negative polarity is correct. The actual voltage level is not important,

only the polarity. It is not uncommon for modems and terminals to work correctly with 3 volt signals, or even 3 volts positive and 0 volts negative. This is something to be aware of, however, as it explains why certain devices do not operate properly with other devices even though they appear to have adequate voltage levels.

The RS-232 standard also provides for short circuit protection on all pins, and it is very difficult to tie any signals together that will cause damage. However, incorrectly wiring a device will definitely keep it from operating correctly. Determining the correct way to wire is sometimes a bit of a mystery, mostly because so many devices do not properly use the RS-232 standard.

Note that the RS-232 standard applies to physical connections only and has no affect on the type of data transmission used. The baud rates, error checking, protocols, and other software related functions are not within the scope of the standard. This means that just because two devices have RS-232 ports, they may not be able to communicate at all. The actual communications takes place via a set of rules called a *protocol*. TEST's systems use a protocol called TSP for TEST SCADA Protocol which is explained in section 12.

10.3 RS-232 CONNECTORS AND SIGNALS

The traditional connector for RS-232 devices is a DB-25. This connector has more than enough pins to carry the needed signals, and most of the signals are not even needed any more in modern computers. However, most devices still use the 25 pin connector for compatibility reasons even though as few as 2 pins are actually used. IBM devised a 9 pin RS-232 subset for the AT computer that has become fairly common, although most installations simply use a 9 pin to 25 pin converter so that the port can actually be used with existing equipment and cables.

The connectors come in a male and female variety, one with pins in it and one with holes in it. Unfortunately, the one with the pins (the male) actually surrounds the mating connector, so it is sometimes incorrectly called the female. To avoid this, it is recommended that the connectors be referred to as "pins" and "holes" to avoid confusion.

Because RS-232 is a two way transmission, it is not easy to identify either side of the link as the sender or receiver. The standard uses the term DCE (data communications equipment) and DTE (data terminal equipment). This made it easy to tell which component was being referenced because the computer or terminal was the terminal equipment (DTE) while the modem or whatever was the communications equipment (DCE). That is fine until two computers are connected. Then, which is DTE or DCE becomes more complicated. The best way to avoid any problems is to know the pinouts of the equipment, understand the RS-232 standard, and make a cable.

The 2 and 3 letter codes in the tables below are the common abbreviations for the important RS-232 signals. These codes have historical roots in the communications industry that sometimes hide the current function of the device. It is important to understand the function of each signal in each application because a single wrong or missing connection can keep the entire system from operating properly. One important feature of each signal is whether it is transmitted or received from a particular device. One unit's transmit is another unit's receive. For example, modem type devices send a DCD that is received by computer type devices. This can get very confusing when one computer connects to another because there is no source of DCD unless a wiring change is made. These types of cables are called "null modems" because they effectively replace the functions normally supplied by a pair of modems and a communications system.

DB-25 PC PIN	DESCRIPTION			DB-25 MODEM PIN
1	---------- Safety ground		----------	1
2	---------->	TD Transmit Data	---------->	2
3	<----------	RD Receive Data	<----------	3
4	---------->	RTS Request to Send	---------->	4
5	<----------	CTS Clear to Send	<----------	5
6	<----------	DSR Data Set Ready	<----------	6
7	---------- SG Signal Ground		----------	7
8	<----------	DCD Data Carrier Det.	<----------	8
20	---------->	DTR Data Term. Ready	---------->	20
22	<----------	RI Ring Indicator	<----------	22

SIMPLIFIED DB-25 RS-232 PINOUT

DB-9 PC PIN	DESCRIPTION			DB-25 MODEM PIN
1	<----------	DCD Data Carrier Det.	<----------	8
2	<----------	RD Receive Data	<----------	3
3	---------->	TD Transmit Data	---------->	2
4	---------->	DTR Data Term. Ready	---------->	20
5	---------- SG Signal Ground		----------	7
6	<----------	DSR Data Set Ready	<----------	6
7	---------->	RTS Request to Send	---------->	4
8	<----------	CTS Clear to Send	<----------	5
9	<----------	RI Ring Indicator	<----------	22

DB-9 (IBM-AT) RS-232 PINOUT

TD Transmit Data, also called SD for Send Data. This term refers to data coming out of data terminal equipment (DTE) probably into a modem (DCE). When two computers are connected directly to one another, their TD and RD lines must be crossed with a null-modem device.

RD Receive Data. This term refers to data coming into a computer (DTE) from data communications equipment (DCE). When two computers are connected directly to one another, their RD and TD lines must be crossed with a null-modem device.

RTS Request to Send. This control signal from a computer (DTE) is used to activate some types of modems. In a direct connect or multi-drop systems, this signal tells the modem to turn on its carrier signal. On other systems, the RTS is not actually required. *Note: TEST's Modem Monitor device uses the RTS line to activate the watchdog timer function. This can be changed with the MONITOR command to use the DCD line, but this requires a jumper change to the Modem Monitor circuit.*

CTS Clear to Send. This signal from a modem device indicates that the modem is ready to receive data from the computer. This is not used by any of the TEST communications systems.

DSR Data Set Ready. Another relic signal from a modem that indicates that the unit is ready to receive commands or data. Not used in TEST systems.

DCD Data Carrier Detect (also written as CD). This signal originally meant that a tone carrier was being received by a modem device. This would indicate that the modem is "hearing" another unit with an acceptable signal and that data received would be valid. Modern communications devices can generate a DCD signal for a variety of other reasons that generally mean that the unit is in communication to another unit regardless of the presence of an actual tone carrier. The TEST SCADA software uses the DCD signal to control on-line status during serial communications.

DTR Data Terminal Ready. This control signal from a computer is used to enable a modem type device. In practical use, the signal is also used to reset smart devices like Hayes modems and Packet Controllers. This signal is toggled off then on during disconnect actions as a hardware method of resetting the modems connected to the computer via the serial port. Not all modems react to this signal, and some act in various ways that depend on jumper or software setting. For example, 2400 baud Hayes modems use the &D command to set the response to the DTR transition.

TEST uses the RS-232 signals in fairly traditional ways for most functions. The most unusual application is the use of RTS or DTR as a control line for TEST's Modem Monitor device. The software toggles one of these lines once per second to activate the watchdog timer in the Modem Monitor. If the line does not toggle within a specified time period, the Modem Monitor will reset the computer. See the MONITOR command and the section on the Modem Monitor for more information.

The RTS line is also used as a special control signal on multi-drop systems. The RTS line is used to control the carrier output of a modem. The software will turn this signal on prior to transmission and will turn it off when the transmission is complete. Therefore, the watchdog timer in the Modem monitor may have to be moved to the DTR pin if it is likely that a transmission will not occur within the timeout period of the Modem Monitor.

In multi-drop mode, the system will monitor the DCD line to make sure the communications line is quiet before turning on the data carrier. If the DCD is clear, the software provides for a delay between the time the RTS line is turned on and transmission starts. This allows modems and radios to get their carriers turned on before the actual data starts down the line. At the end of the data burst, the RTS line is kept active for a short time before it is again turned off.

Direct connect systems can be hooked up with a simple 5 wire null-modem cable. This simply crosses the TD and RD lines from each of the computers and also switches the RTS and DCD lines to provide flow control. When the direct connect system "dials" out, it simply raises its RTS line and keeps it on for the duration of the session. The other unit will "see" this signal at its DCD input which will cause it to turn its own RTS line as a response. Therefore, direct connect systems use their RTS line to activate the DCD input of the other computer. The fifth wire is used for the SG Signal Ground, Pin 7 on the DB-25. Therefore, a null modem hookup on a 25 pin connector would use pins 2,3,4,7, and 8. Pins 2 and 3 are crossed, and pins 4 and 8 are crossed. Pin 7 is a straight through connection.

10.4 DATA MODEMS

It is possible to connect two devices directly using serial ports and the RS-232 connectors. This is fine for local connections to terminals and printers. However, connecting devices over a longer distance requires some sort of data carrier device, called a modem. Modem is short for MODulator - DEModulator. This refers to the device's ability to take a true or false signal (1 or 0) and transmit it by modulating (changing) a tone signal. The Demodulator part of the device detects the amount of signal change and interprets it as a 1 or 0 on the receiving end. This allows ones and zeros to be transmitted over any link that can carry sound.

The RTU/SCADA system uses many types of modems. Some are phone based and are designed around the HAYES smart modem standard. These are used for all types of phones, either hardwired or radio based. Another type of modem is called a PACKET RADIO CONTROLLER, or PRC. This device is intended for operation over radio links. The system will also work with various types of simple modems and multi-drop modems that work with various other communications systems.

TEST's system will also work with PC based Network systems. The system is similar to a multi-drop system except that data is exchanged over the network rather than over a communications channel. The details of network operation are explained in a separate document and will not be covered in detail here.

All modems do basically the same thing, which is change data into sound and then back again. The difference between the different modem designs is the type of physical output to the communications link. Some are designed for telephone type connections (RJ11 Jacks), while the other has separate audio input and output for connection to the radio. Still others are designed for dedicated wire pairs or microwave circuits.

10.5 RTU COMMUNICATIONS MEDIA

The RTU/SCADA program is designed to allow operations from the local PC keyboard and CRT as well as via remote communications links. To accomplish this, most functions within the program do not assume anything particular about the link. Any special requirements are taken care of at a higher level within the program.

The communications media types currently supported are:

Local Console	Local PC CRT and Keyboard
Direct Connection	RTU or ANSI terminal hardwired to port
Phone Modem	HAYES modem on serial port
PRC Modem	Packet Radio Controller on serial port
Multi-Drop	Special modems or radios on serial port
Netbios	PC Based Network Communications

Aside from the dialing and hang up type functions, the operation of the program over the modem is identical to hard wired or local console operation. The software is not directly aware that the data is being transmitted with sound instead of electricity. The RTU/SCADA programs, however, do a lot of housekeeping to manipulate the modems in order to make the connections, maintain them, and then disconnect for a variety of reasons. In the case of the hard wired connection, these functions are not needed, although the status of the connection is always monitored.

The SET MEDIA command is used to tell the program the type of link used for each RTU type task. It is normal to place the proper setup commands in the STARTx file for each task so that the proper setting will always be obtained on startup. The commands can also be repeated in the BYE file for each task to insure that any changes made while online are fixed when a user logs off.

10.6 MODEM ON-LINE STATUS

The RTU/SCADA program is particularly sensitive to the on-line status of the communications device and terminal. This is the result of numerous problems with actual installations where the communications channel is unreliable and extremely noisy. The basic means of detecting an on-line state is via pin 8 of the RS-232 line, the DATA CARRIER DETECT (DCD). *The DCD input for any type of serial communications link must accurately reflect the state of the connection.*

The CD or DCD line has traditionally meant that the modem device was sensing the audio tone called the "carrier" that was used to carry the data tones. The variance from the carrier is what determines the one or zero state of the line. If no carrier is being detected, the modem is off-line and any data coming from it must be ignored. This is still true for many modem types that operate in a point-to-point mode with only two devices at a time.

Because of the variety of communications links possible with the program, the DCD line is always used as the common means of determining that the link is valid. In the case of the directly connected terminal, the DCD line must be jumped to another pin on the port that is in the same state as a true DCD (i.e. +12v). This will tell the program that the terminal is connected at all times.

For a HAYES type modem, the DCD must reflect a connected state to another modem. As long as the HAYES determines that there is a carrier, or that a long enough break in the carrier has not occurred, then the DCD will be true. If the carrier is not present, then DCD will be false and the program will process a disconnect. A setting, either by switch or by command to the modem, is *necessary in order for the DCD to function in this way.* A constantly active DCD will cause the program to operate as if it were always connected, which is incorrect. This will prevent dialouts and cause the program to loop waiting for data that will never come. Refer to the modem documentation for more information about modems and how to set the DCD for proper operation. *Tip: See the &C command in your Hayes compatible modem reference book.*

The packet radio controller (PRC) is a little more complicated because of the "half-duplex" nature of radio transmissions. The PRC does not send out a constant carrier tone like the HAYES modem does. In the case of the PRC, the DCD status represents a logical connection to another PRC that is determined by radio tone handshaking by the modems themselves. One PRC modem calls another, they exchange codes, and then the DCD is established. Until the PRCs disconnect, the DCD will be true even though no actual carrier is being transmitted. The disconnect will occur after a timeout or if the computer commands the PRC to "hang up". Note that a jumper or other setting inside the PRC must be set properly in order for the DCD to provide this function. Not all PRC units offer this type of DCD function, so check with the manufacturer if you are using a non-standard unit with a TEST system.

Multi-drop modems are different from direct-connect devices because the carrier signal is present only in brief spurts of data. Therefore, multi-drop configurations do not use the presence of the DCD as a mechanism to detect an on-line state. The multi-drop system uses a logical rather than physical on-line , much like the Packet Controller. The software keeps track of the on-line status based on the characters received and various timeouts. Therefore, tasks using multi-drop modems will not use the DCD line to determine their on-line state.

10.7 HAYES MODEMS

The HAYES standard refers to modems that can do many functions in response to a special command sequence. The standard is used by most PC modems, although many modems have non-standard functions that are hardware dependent. The modems are often referred to as using the standard "AT" command set which identifies the modem as a Hayes type modem without having to use the actual word Hayes in the specifications.

The commands are centered on things like phone dialing, timeout detection, call progress monitoring, and other communications functions. The processing of these functions is done within the modem itself using a small microprocessor. The commands go into the modem over the same serial line that the data uses. So, the modem watches for a special sequence of pauses, plus signs, a pause, and the letters "AT" (for attention) to switch from data mode to command mode.

The ability of the modem to do the dialing and timing functions eases the burden on the main PC processor. However, the modem must be correctly set up in order to function with the software. Although the means of setup are slightly different for most HAYES modems, the basic setting requirements for most installations are:

1. Carrier Detect must follow the true carrier on the line and must <u>not</u> be present at all times. This is set via a switch on 1200 baud modems and by the &C command on all other Hayes compatible units.

2. Tone dialing (depends on the unit). Override with P if pulse dialing required instead of touch tone dialing.

3. Answer on 2 rings (S0=2).

4. Lost carrier delay of 1 second or mode (S10).

5. DTMF timing set fairly slow (S11) for cellular and radio phones. Fast dialing on land lines is usually OK.

There may be other settings particular to each installation and modem that will require setup. The program's HAYES command can be used to send these commands to the modem at startup or each time that a BYEx file is processed. This is recommended to insure that the modem is always properly setup after each communications session.

The HAYES modem can be accessed at the RTU or HOST by using the ATTACH command. This will connect the local CRT/KEYBOARD to the serial port going to the modem. Commands can be sent and their responses seen on the local CRT. Note that the modem must be placed on "QUIET OFF" so that it will send responses, and "ECHO ON" so that the user's keystrokes can be seen. This will require a "Q0 E1" command to be sent to the modem. Refer to the modem manual for further information.

The phone number in a Hayes dialing string can contain special characters that are often helpful in working with real-world telephone problems. These special codes go right in there with the actual digits and instruct the modem to do special things during the dialing process. The most common codes are listed below.

CODE	FUNCTION
P	Switch to Pulse style dialing
T	Switch to Touch Tone dialing
W	Wait for a dialtone before proceeding
,	Programmed delay (normally 1 second, adjustable)
/	Wait 1/8th second
!	Flash
@	Wait for quiet before proceeding

These codes are used to handle special dialing problems such as getting an outside line, long distance codes, cellular tricks, and other real-world telephone problems. Consider the dialing string P9,T555-1212,454. This character sequence will switch to pulse dialing and dial 9 perhaps to get an outside line on an older switchboard. The comma after the 9 will cause a wait of 1 second which allows time for the internal switchboard to access the outside line. Then the modem switches to Tone dialing and dials the number 555-1212. At the end of the phone number, the modem will wait another second for a long-distance service to kick in. Then, the string will send the tones for the digits 454 which could be for a special long distance access code.

Obviously this was a made up example but is typical of the dialing problems that are encountered in actual installations. The most common codes are the P and T to control the dialing method and the comma to put waits into the sequence.

There are also several "S" registers internal to a Hayes modem that control timing and other parameters. Most Hayes compatible modems support a basic set of these registers and others have many special ones unique to that particular model of modem. Consult the manual for exact details on each modem. The S registers needed in most applications are listed below.

S0	Number of rings before answering, default is 0.
S6	Sets seconds to wait (max) for a dialtone (default 2)
S7	Sets seconds to wait for carrier on callout (default 30)
S8	Sets seconds for delay for each comma in phone number
S9	Carrier response time in 1/10 seconds (default 6)
S10	Delay from loss of carrier to hangup (default 14/10 secs)
S11	Milliseconds between touch tone dial codes (70 ms)

These S registers are set in the modem by using the HAYES command, normally during startup and in the BYE file for each task. For example, to set a modem to answer on 3 rings and use 150ms between touch tones we would use

```
HAYES S0=3  S11=150
```

Hayes modems are very complex devices. The manual should be consulted when troubleshooting any communications problems that may be related to behavior of the modem.

10.8 PACKET RADIO CONTROLLERS

The PRC is similar to the HAYES modem in that it contains a small microprocessor that handles many timing and other communications functions. It goes beyond the HAYES modem in that it also handles message routing and error checking. Rather than send data directly as it comes in, the PRC gathers up a group of data called a PACKET and sends it as a chunk to another modem. The packet has the address of the sender and receiver as well as error checking codes to insure that the information is properly routed and received. The PRC receiving the message sends acknowledgement codes to the sender for each packet.

There is obviously a lot of work involved in all this processing, but the PRC takes care of it automatically. Rather than dialing a phone, the PRC sends out a connect request to another PRC. If the connection is made, the PRC uses the DCD line on the RS-232 port to indicate a connection. Data can then be sent by the computer without any concern over the additional processing being provided by the PRC. Problems like data errors and disconnects are handled by the PRC which will notify the computer by killing the DCD signal.

The PRC must be set properly for each installation in order for it to function correctly. The complexity of the radio transmission requires detailed information on the radio's audio input and output circuits that will be connected to the PRC. Also, the "key up" times and squelch settings of the radio must be adjusted and accounted for with various settings within the PRC. The most important settings are:

MYCALL	The radio address of the PRC, normally set to the RTU's ID code.
TXDELAY	Transmission key up delay, or the time between the PRC's push to talk and the actual transmit.
DCD	Setting that sets DCD to indicate a logical connection to another unit.

The PRC can be diagnosed and setup by the RTU or HOST computer by using the ATTACH command. This will let the local CRT/KEYBOARD be used to send and receive characters directly to the PRC. Note that the PRC must have its ECHO ON and STATUS ON in order to see keystrokes and responses from the PRC. Refer to the PRC documentation for further details.

10.9 DISPLAY TERMINAL TYPES

Different physical display types require different control codes to do things like clear the screen and position the cursor. In the past, there were hundreds of different terminal styles that made if difficult to write software to work on all of them. This has been eased in recent years by the adoption of an ANSI terminal standard pioneered by Digital Equipment Corp. (DEC). In addition to the PC's CRT and keyboard, the RTU/SCADA program supports ANSI terminals either remotely via modem or directly connected to the computer. *Note: The previous support for Televideo, Adds, Hazeltine, and other stand-alone CRT devices has been dropped in all software issued after June of 1990.*

The most basic terminal type is the local CRT and keyboard. Although not really a terminal device, the program treats it as such in order to avoid making a special case for the local user. A few conveniences (like function keys and colors) are supported for the local user. There can only be a single local CRT on each RTU computer, although additional terminals can be locally installed using the serial ports. These terminals, however, are treated as remote terminals by the program.

Remote terminals must conform to the basic levels of the ANSI terminal standard. This can be accomplished with a stand-alone ANSI terminal, or by using a PC emulating an ANSI terminal. The RTU/SCADA program has this emulation capability via the ATTACH and CALL commands, although most communications programs have more elaborate emulation capabilities. PROCOMM is an excellent software product that can be used to connect to an RTU for diagnostic and display purposes.

SCADAWARE also contains built-in support for QTERM hand-held LCD display and keyboard units. The CRT type designation for these is QT2. SCADAWARE will generate the proper code sequences to position the cursor, highlight, and clear the screen of these units. The QTERM devices are used for very simple I/O only, and cannot be used as a system console to program or configure the computer.

In summary, the CRT types currently accessed with SET CRT xxx are

CGA	Normal 80x25 color mode
MONO	Normal 80x25 Monochrome Mode
EGA	Small character 80x43 line EGA mode
VGA	Small character 80x50 VGA Mode

Ansi	Standard Ansi Terminal, 80x24
QT2	Qterm 4 line x 20 character mode

10.10 TEST INC. MODEM MONITOR

The Modem Monitor is a TEST INC. designed and manufactured device intended to assist in unattended operations. Although not a communications device itself, it is included here because it ties into the RS-232 line to help monitor system activity.

The monitor is intended to allow an automatic or manual reset of the computer if a malfunction occurs in the processor. The operation is slightly different for radio and modem configurations. The functions of the device include:

1. Monitor the RTS output from the PC acting as a watchdog timer that must toggle periodically to indicate normal operation. A timeout on the watchdog causes a reset.

2. In phone modem systems, count the number of ring signal transitions and reset the computer if a carrier detect is not received within the count.

3. In Radio systems, count the number of CD transitions and reset if data from the RTU is not detected within the count.

4. In phone modem systems, while a CD is detected, time a "break signal" from the other computer and reset the local computer if the time is exceeded.

5. In radio systems, time the length of the CD and reset if the specified time limit is exceeded.

The device is intentionally simple in design (i.e. no processors, memory, etc.) to provide the most reliable operation in adverse conditions. It can operate from +24, +12, or +5 volts. Power consumption is approximately 20 ma. The signals that are monitored on the RS-232 line can be changed to accommodate special installations, although a PC type serial port is assumed. The options on the board are set by soldered wires or jumpers making adjustments in the field possible by service personnel.

The design allows for the monitor to be spliced directly into the ribbon cable running between the PC and the modem. The only other connections necessary are the power supply and the computer reset signal. All of these are brought out on a single cable that plugs into a mating DB-25 installed in the modem cable. If a problem is suspected with the modem monitor, it can be easily unplugged to remove it from the system. The only consequence of removing it is that it can no longer function as a monitor, but the rest of the system remains unchanged.

The RTS line used as the watchdog signal from the PC only operates while SCADAWARE is running. If the RTU or HOST is being used for other purposes the monitor must be disabled or unplugged in order to avoid periodic resets.

The watchdog action on the RTS line may affect the proper operation of the modem device connected to the serial port. If this is the case, pin 4 of the modem connector can usually be pulled out to avoid confusing the modem. Most modems do not need to use RTS, but they may monitor it to detect the on-line status of the computer. The toggle caused by the watchdog may make the modem think that the computer is having problems and cause an unintended disconnect.

11 COMM LINKS AND CALLOUT GROUPS

11.1 SCADA COMMUNICATIONS

Most SCADA systems provide some means of transferring data from one system to another over a communications link. Some less sophisticated systems assume that the problems of phone numbers, protocols, and other nasty communication details are taken care of outside the SCADA system itself. This is often the case in PLC and Laboratory type off-the-shelf packages that assume a human operator or existing comm link. The TEST SCADA System, however, has extensive internal support for linking a number of units over a variety of dial-up and continuous communication links.

11.2 COMMUNICATION LINKS

Under the TEST SCADA system, a comm link is a relationship between an RTU task and a specific phone number (or radio call sign). This is the basis of the link - a phone number and a means to dial out with it and communicate. Additional parameters like retry counts and timeouts are also associated with the link. *A link does not do the callout by itself.* The link contains the setups necessary for one of the communications tasks to do the actual callout using a serial or network communications link.

The program's comm links are a powerful means of controlling how the computer will connect with remote systems. Simple systems will have a single link to cause a callout to a single remote unit. More complex systems have numerous remotes, each with its own link, as well as alternate links and paths to other host systems. The system allows a variety of communications methods to be used on a single system because each link can have its own set of operating parameters.

Because a link is simply a set of parameters, any number of links may share the same RTU task or phone number. For example, one link may use TASK 1 to communicate with a cellular phone connected to serial port 1. Another link may use TASK 2 to reach the same number over a microwave phone system attached to COMM PORT 2. Also, the same task may be used to reach several phone numbers by assigning them to different links. This may be useful if different remote systems must be called using the same phone line. *Remember, the link is just a setup, not an actual communications path.* A comm task does the communicating. The link simply tells the task what to do and how to do it.

The system defaults to 4 separate links but any reasonable number of links is allowed. To change the number of links for a system use the keyword LINKS in the main configuration (DAT) file followed by the number of links desired. The number of links available is set during program startup and cannot be changed while the program is running. However, links can be modified at any time and any changes take affect immediately.

A link can be in only one of three possible states: IDLE, ACTIVE, and FAILED. All links start out as idle and are activated by a number of methods within the system. An active link remains that way until it is either reset (after a successful download), or the link fails. A link can also go from active to failed if it is not reset after a specified number of retrys has occurred.

Links become active for a variety of reasons. The most common reasons are an alarm condition at an RTU or a poll request at a Host. The reason for the Active state is not important to the link. Once it becomes active, it will go through the same steps regardless of the source of activation.

All communications tasks check the link table to see if any links assigned to them need "servicing." This would mean that they are active and their various delays have timed out. The task responsible for servicing the link uses the information in it to attempt a dialout. The dialout can be on phone, radio, wire, or whatever. The term dialout is used in the TEST system to indicate any type of communications attempt.

If the dialout attempt results in a connection, data is usually transmitted in one direction or another to satisfy the reason for the link's active status. Part of the transfer process must involve a link reset action of

some form to force the link status back to Idle. The is done with a LINK RESET command or as part of the time stamp process at the end of a download.

If, however, a connection is not made or a call can not successfully complete its transfer due to a communications problem, the call is terminated and the link remains in the active state. After a programmed timeout the link will again become eligible for servicing. If the later call is successful the link will be returned to the idle state. Otherwise, the link will remain active and wait for another timeout before attempting another call out. When the number of unsuccessful call out attempts has reached its limit, the link will be set to the failed state. It will remain in this state until it is reset.

Note: A failed link remains failed until reset. This can cause a loss of alarms because subsequent alarms that affect an already failed link will not cause a link reset. The link must be reset by some external process such as a daily Agenda activity or a manual poll request.

A LINK ALL RETRY command can be used that will return all failed links to the active state with a fresh set of timeouts and retrys. This can be used in a daily agenda file to restart any links that had failed for any reason but it will not affect any links that are already active and processing normally.

When a link callout is successful, the system automatically trys to locate a special LINKx.RTU file that will be processed on the unit processing the link. So, a unit making a call with link number 3 will look for a file called LINK3.RTU (or a procedure named LINK3) and will process it if found. With this capability, the system can be programmed to do different things with different links. One link may request a download from an RTU, while another may relay data from one host to another. Because each link is treated separately, individual responses can be easily programmed for various requirements by using different links and their associated link files.

SCADAWARE links can also become "Blocked" from any further activation after exceeding a specified number of callout attempts. The purpose of this feature is to prevent high callout counts on cellular phone systems. Once blocked, nothing can activate a link until it is cleared with a LINK x ZERO command. This ZERO function can be done at any time, with the DAILY procedure being the most likely place.

11.3 COMM PORT USAGE

Any RTU task in the system that communicates with either a person or another unit must have some sort of communications channel associated with it. For most communications the PC type systems use serial communications channels, called serial COM ports, and the local CRT/keyboard. Each channel has a number associated with it. 0 is the local CRT and the serial channels are numbered consecutively beginning with 1. Thus, COM1 is the first serial port, COM2 is the second, and so on. The actual IBM specification only allows for COM1 and COM2 on a PC, but many developers (including TEST) have extended this to allow for additional ports.

To keep things simple, each RTU task is assumed to use the com port that is the same number as the task. Thus, the local task 0 uses the local CRT (COM0), task 1 uses COM1, task 2 uses COM2, etc. If at all possible keep this relationship as it is. If necessary, however, this can be changed by using the SET PORT command.

Each serial port in the computer can only connect to a single communications device. This is normally a modem or PRC as discussed earlier in the section on communications. So, if COM1 is attached to a modem, task 1 should be configured (via the SET MEDIA command) for a phone modem. All communications links that use a phone will make use of task 1 to do the dialing and necessary command functions. If a radio is also used on the system, it must be connected to the computer over a serial port with a PRC. The PRC and phone modem cannot be on the same serial port. So, an additional task, Task 2, can be used to handle the PRC. Any links that require radio communications will then use task 2 to do the processing.

11.4 COMM LINK FEATURES

Each comm link has a set of parameters that can be programmed using the PROG or CONFIG command. An alternate programming method using a simple command line is also available for programming within command files. See section 11.5 for explanation of the LINK command.

```
ÕÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¸
³                                LINK 2  Of 10
      ³
ÔÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍ¾


Link Description  Radio to Kiss B

Number of Task Using This Link  1

Phone Number or Radio Call Sign  RTU1

Seconds to Wait For a Connect Before Giving Up  5

Maximum Number of Tries to Call Out When an Attempt Fails  3

Number of Tries to Rapidly Call Out When an Attempt Fails  2

Seconds to Wait Between Call Out Attempts After Exceeding Rapid Tries  60

Maximum Number of Tries to Call Out Without Resetting Counter to Zero  0

Does This Unit Act as a Host When Callin♦ Out?  Y

Alternate Link  -1






ÕÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¸
³  <F2> Accept  •  <PgUp> Previous Link  •  <PgDn> Next Link  •  <Esc>
Cancel  ³
ÔÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍ¾
```

Each link is a data structure with the following features:

LINK NUMBER: Each link has a unique number. By default, only 4 link's are allocated at startup unless a LINKS line appears in the main configuration file. The link's number is the means to identify it in any reference. Each link has the following individual settings:

1. LINK DESCRIPTION Each link has associated with is a text field that is used to describe the link.

2. TASK NUMBER: Each link must be controlled by an RTU task, and only one RTU task. Assigning the task number places the responsibility for servicing the link's callout needs on the specified task. Assigning a link to a task that does not have a serial communications port associated with it will not work very well.

3. PHONE NUMBER: Although called a phone number, this is just a sequence of characters that will be sent to the modem or PRC when a callout is required. This can be either a phone number or a radio call sign. The system will take care of the details required to get the phone number or call sign transmitted to the device in the proper format. Only the actual phone number is required here. Do not put any

HAYES or PRC dialing codes in this string unless they are in addition to the codes normally required for a dialout because the program will automatically add them when the string is sent to the device.

4. CALLOUT GIVEUP SECS: When a task starts a callout, it picks up a seconds count from the link description and starts a down counter with that value. The task will wait until the timer gets to zero before giving up on the callout. If no DCD is detected, the task terminates the callout and starts the between callout delay timer for the link running as described above. The proper setting for the give up will depend on the speed that the comm channel can place a call, typically between 15 and 120 seconds. *Note: The modem device may require special programming in order to make it attempt a longer than normal callout. For example, the Hayes modems need to have their S7 register set if the callout wait is longer than 30 seconds.*

5. MAXIMUM CALLOUT TRYS: When a callout is first initiated for a given link, a counter is cleared that will track how many times the callout is attempted. Each call increments the counter, and callouts will cease if the counter exceeds this maximum. This can be used to prevent a system from tying up a comm line if for some reason the other end is not responding. The setting will depend on how reliable the system is, the cost of each callout, and the available power to operate the communications system.

6. RAPID CALLOUT TRYS: The rapid callouts occur during the initial attempt for the system to establish a link. SCADAWARE will make several quick attempts (as determined by this parameter) in which it trys to connect, disconnects, and rapidy trys again. This is useful in certain shared comm systems, such as Cellular Phones, where a rapid retry is desirable to try and get a line in a busy system. This parameter determines how many quick trys the system should attempt before reverting to the normal try and wait sequence.

7. NORMAL CALLOUT DELAY: This determines the number of seconds to wait between callout attempts after the quick trys are exhausted. The timer starts counting down as soon as the system gives up on the current callout. A callout on the link will not be attempted until this timer reaches 0. It is possible for the timer to sit at 0 for a while before the link is serviced if the responsible task is busy servicing other links that became available sooner. A short delay is 15 seconds, while a long delay is about 1200 seconds (20 minutes). Note that this delay is not used during the quick callout period.

8. MAX TRYS WITHOUT RESETTING COUNTER TO 0: Each link has a counter which tracks how many times the link has been attempted. This counter is reset only by a LINK x ZERO command. If the setting for this parameter is non 0, then the link will become "blocked" when the callout counter exceeds the parameter. This prevents endless calling from a link which, in the case of cellular phones, can create a huge phone bill. Setting this to 10, for example, will limit the callout attempts to 10 until the internal counter is zeroed. The Zero action can be done at any time, but is usually done in the Daily file for each system.

9. IS THIS A HOST: This parameter determines a system's position as either a HOST or an RTU during link processing. Normally, there is no difference between an RTU and a HOST for any system operation (other than real I/O). The term HOST and RTU refers only to the logical use of the system, not its physical make-up.

10. ALTERNATE LINK: In the event that a link can not successfully complete a callout and the maximum number of tries is exceeded, the link will be set to the failed state. At that time the alternate link will be activated if one is specified. A value of -1 is used to indicate that no alternate link is specified. If the alternate link has already failed, it will not be reactivated as part of another link's failure action.

The "IS THIS A HOST" parameter will allow the system to control its reaction during a callout without the use of a LINK file. Although the LINK file is still supported, it is no longer required. Most link files simply contain a line containing READ DOWNLOAD or BLOCK READ DOWNLOAD. Now, the system will generate this message automatically depending on the setting associated with the current link. This only affects callouts on a link. Incoming calls still depend on the caller (who is processing a callout link) to control the communications.

The older LINK file method will still work as before. The change simply eliminates the need for the LINK files in most systems because the action previously done with the LINK file will now be done automatically. Upon calling out, the sequence is now as follows:

1 - Dial the number associated with the link.
2 - Wait the give up seconds for a connect.
3 - After connect, look for a LINK file.
4 - If file existed, process it.
5 - If no file, then:
    If an RTU process READ DOWNLOAD
    If a HOST process BLOCK READ DOWNLOAD

This parameter is also being used to handle collisions between a HOST and an RTU that call one another at the same time. Previously, a dead-lock occurred where both systems were processing LINK files. Typically, the HOST was processing BLOCK READ DOWNLOAD while the RTU was already processing the first line of the DOWNLOAD. This resulted in both systems waiting for an ACK to a command and the systems would timeout and terminate communications.

Now, when a system is waiting for a numbered acknowledgement, and an ACK number 0 is received, a system operating as an RTU will terminate file processing but remain on-line. This should result in the other system, which is probably acting as a HOST, to resend the line. This will allow a priority relationship where the HOST has control of the session. The HOST will continue trying to send the command line (after the ACK wait timeout), while the RTU will suspend file processing and wait to receive commands rather than send them.

This will only work properly if both systems have been correctly set up as either HOST or RTU. If two RTUs call each other, then both will suspend operation and a timeout will occur. If both are set up as HOSTS, then each will continue trying to get a command through and a similar timeout will occur.

11.5 PROGRAMMING A LINK

Links can be programmed with the CONFIG LINK x command where x is the link number. When using this command, all link settings are displayed at once and the user can move around the screen and edit the settings in any order desired. When configuring is complete, the user can press ESC to cancel changes or press F2 to accept the changes.

Links can be interactively programmed with the PROG LINK x command, where x is the link number. When using this command, the user is prompted with the current settings and modifications can be made. If a current setting does not need to be changed the user can simply hit the [ENTER] key to accept the existing value. *NOTE: The PROG command in SCADAWARE Lite can only be used if the main configuration (DAT) file contains the line LOAD PROGRAM.*

Note that the CONFIG command only works on the local CRT. To program links remotely over a serial line, the PROG command must be used. With both commands, the changes will take effect immediately but are not permanently saved to disk until a SAVE LINK command is issued.

A LINK can also be programmed via a command line which begins with the keyword LINK. The parameters which follow the keyword are:

1. Link number
2. Phone number or radio call sign
3. Task to process the links
4. Max Retry count
5. Callout give up seconds
6. Between call delay seconds
7. Is this a Host?
8. Link description (any text)

9. Alternate link to use if this one fails
10. Rapid Trys
11. Max calls without Zero

Any parameters that are not available can be left blank (between commas) which will allow the existing value to remain unchanged. A typical line would look like:

```
LINK,3,1,555-1212,5,45,120,No,Phone to Host,-1, 4, 30
```

This programs link number 3, which uses task 1 (and probably serial port 1), with phone number 555-1212. The retry count is 5, and the callout give up is 45 seconds. After missing a callout, the system will wait at least 120 seconds before trying another callout. This system is not a Host. The link is used to call the host and does not have an alternate link.

11.6 DESIGNING A COMM LINK

Setting up a link requires some knowledge of the physical channel being used. One thing that all communications channels have in common is that they are all different. How the channel will be used in each installation depends on several factors, of which the chief ones are:

1. AVAILABILITY: Is the comm channel used exclusively by the SCADA system, or is it shared with other users (such as people, fax machines, other computers)? If not, then retry counts and timeouts can be set to allow the system to "hog" the channel as much as it wants. If it is a shared medium, then the link can be set to allow time between calls as well as a maximum retry count to prevent the SCADA system from dominating the comm channel.
2. CALLOUT RESPONSE TIME: Some systems are slower than other similar systems when placing calls (or connecting via PRC). This is due to the numerous telephone configurations that are in place in the world. Some systems place a call almost immediately, while others may take up to a minute to get the first ring back from the other end. Some experimentation is required to find the optimum give up time for each link.

3. RELIABILITY: No communications channel is perfectly reliable, and the ones in remote locations where TEST's SCADA systems are installed are among the worst there is. If a fairly reliable system is in place, then the retry count can be set fairly low (2 or 3). This will allow processing of a communications link failure as soon as possible. If the reliability is not too good, then setting the retry count higher will provide more chances to get a call through, but will delay processing of the CommFail file for a longer time.

11.7 ACTIVATING A LINK

A link is in the active state when it needs servicing and has not exceed its callout retry count. A link can become active for a number of reasons. The most common method is when a channel enters a condition that requires a callout. Using a callout group (explained below), each channel can specify which links are to be activated when a callout is required.

A link can also be activated when a request is made on an RTU to communicate, usually by pressing the F5 key while the desired RTU is selected. The older POLL NOW command may also be used to accomplish this function. Multidrop systems may use the ALERT command from an RTU needing servicing (normally a Type 2000 or 2200). The ALERT will indicate which RTU wants attention, and the link which has that RTU ID as the call sign will be activated so that the Host can perform a download from the appropriate RTU.

If an alternate link if specified within another link's setup, it will become activated if the primary link fails. The alternate link will proceed with its own set of parameters.

11.8 CLEARING A LINK

A link that has been activated will remain in the active state until it is cleared or until it exceeds its retry count. Because all data transfers are programmed with command files, it is easy to place the appropriate LINK RESET command in the proper place to ensure that the link is not reset until the data transfer has been properly completed. The LINK RESET will normally be associated with the download activity, which can be initiated from either the calling or answering unit. In any event, the LINK RESET should not occur until all required data has been transferred.

When the system operates in block transfer modes (i.e. computer to computer links), each message is verified with an error detection and retry scheme. If a message cannot be sent properly, the system automatically disconnects the comm line. The online status of the comm line can be monitored with the SET ONLINE command to prevent further file processing if the line is lost. Thus, a LINK RESET placed at the end of such a transfer file will only be executed if all previous commands were sent successfully, indicating a proper download.

Active links which are being processed by a task can also be cleared with the @ option in a data line, or with the special @ENDS option in a data line. Refer to the DATA command in the TSP command reference for more information.

11.9 MONITORING LINK STATUS

The status of links can be monitored with the STATUS LINK command (or menu option). This command causes a real-time display of all links showing their status (idle, active, failed, or blocked) as well as the various timers associated with each link.

This screen is helpful in diagnosing link timeouts and retry counts to verify that the system is properly configured. The Link Status screen can also be reached from the main menu under the LINK option.

```
HOST  SYSTEM STATUS 09:16:15

COMUNICATIONS LINK STATUS

LINK  DESCRIPTION          TASK  HOST TRYS  CUR  WAIT  LEFT  GIVUP STATUS
 ALT
1     Radio/Wire to KissA     1  Yes     3    0     5     0      5 Idle

2     Radio to Kiss B         1  Yes     3    0    60     0      5 Idle

3     Radio to Kiss C         1  Yes     3    0    60     0      5 Idle

4     Radio to Ibis A         1  Yes     3    0    60     0      5 Idle

5     Radio to Ibis B         1  Yes     3    0    60     0      5 Idle

6     Link 6                     No      3    0    60     0     60 Idle

7     Link 7                     No      3    0    60     0     60 Idle

8     Link 8                     No      3    0    60     0     60 Idle

9     Link 9                     No      3    0    60     0     60 Idle

10    Link 10                    No      3    0    60     0     60 Idle

TASK  LINK CALLING  ONLINE  TIMER    CMD  RESET  INPUT    ACK

1        0      No      No      0      0    291      0      0









                [ESC] to STOP       [ENTER] to TOGGLE DISPLAY
```

11.10 CALLOUT GROUPS

A callout group is a convenient way to associate a number of comm links that will need to be activated at the same time. A group may consist of a single link, or it may consist of all links, or anywhere in between. Also, a single link may appear in more than one group.

The main purpose of the groups is to allow a simple way of setting callout "priority". Low level alarms that may not warrant a general callout to all remotes can reference a group only containing one link. A more serious alarm may reference a group that contains several links so that numerous calls will be made from the single alarm.

The grouping also allows for simplified revision of callout requirements that may be required due to changes in the operating mode of the facility. For example, it may be desirable to have a unit call a different set of HOST locations on the weekend when normal office personnel are not available. Rather than having to reprogram all the channels, the user can simply reassign the group's link designations.

11.11 CHANNEL PROGRAMMING FOR CALLOUTS

As mentioned, each channel has its own group setting (not its own link setting). A simple system may have all points using the same group that uses a single link. This is typical for a simple RTU reporting to a single HOST. When multiple HOST units are connected to multiple RTUs over multiple communications paths, the use of callout groups and multiple links becomes necessary to determine which points call which units.

Remember that providing a channel with a valid callout group is only part of the requirement for a successful callout. Each point must also be programmed to initiate callouts on abnormal and/or return to normal conditions with the alarm mode setting. The CHANGE command can be used to easily modify a range of channels to cause callouts as needed.

11.12 LINK AND CALLOUT GROUP SETTING STORAGE

For convenience, both the LINK and CALLOUT GROUP settings are stored in the same file, which is usually fairly small. This file is typically named after the system, and has a file type of ".LIN" for link. The lines in this file look like lines that could be typed by the user to setup the links and groups. These lines can be generated with any text editor, but are more commonly generated by a LINK SAVE command.

The settings will have to be reloaded every time the system is started by having a task (task 0 for example) read the file. This is similar to how the channel configurations are loaded. So, a simple "READ RTU.LIN" (or whatever your location name is) command can be placed in the start0.rtu file for automatic processing. Even better, use the line "READ $S.LIN" which will have the program replace the $S with the system name.

LINK data is also stored in the system IMAGE file. If the Image is loaded during startup, then it is not necessary to separately load the LIN file.

11.13 DEFAULT LINK SETUPS

The software provides a default number of Links and Groups. The default values can be overridden with commands in the main configuration (DAT) file that gets processed during startup. Also, the default setting for the Link is to use Task 1 with some standard timeout values. Group 1 is set to be Link 1 only.

What this means is that the normal setup is adequate for a simple RTU-HOST setup where there is only one comm link and only 1 serial port type RTU task. The only additional information required would be the phone number or radio call sign that can be set with the PHONE command.

11.14 EXAMPLE TSP PROCEDURE FOR A COMM LINK

TSP procedures or files are used to do special processing when a link successfully causes a dialout to connect with another unit. These are called "Link files." In the absence of a link file, the system will either perform or request a download according to the "Is this a Host" setting of the current link. Link files can perform many of the functions of a normal download file, but they can also perform special actions such as turning on outputs at the remote before requesting the download.

An example of a Link file that calls an RTU and checks for an output pulse requirement is as follows:

```
; Call an RTU and pulse output if needed prior to download
set online on     ; proceed only if CD is present
if NEEDIT         ; Needit is user defined variable <> 0 for action
 block calc T1=30 ; activate output at RTU
 calc needit = off      ; turn off the local status point
 msg Waiting for Output
 sleep 30         ; wait for output to take affect
endif
block read download  ; proceed with normal download
```

This is a simple example, but it would be useful in an actual application. If there was no Link file present, the dialout would have resulted in the command READ DOWNLOAD being sent automatically to the remote unit when the connection is made. This file overrides the default action and allows the Host to check a status channel named NEEDIT. If needit is true, the Host sends the command to cause the RTU timer to run for 30 seconds, apparently resulting in some sort of shutdown action at the facility. After a short wait, the Link file proceeds with the download as normal by sending the READ DOWNLOAD command at the end. At that point,

the remote will take over control of the session by processing its DOWNLOAD file.

12 TSP - TEST SCADA PROTOCOL

12.1 SCADA PROTOCOL BASICS

TEST's SCADA system, like any other SCADA or telemetry system, uses a precise method of exchanging data and commands among the units connected to the network of devices. These methods are collectively called a *PROTOCOL*, and there are many different ones in existence. Some have evolved from the early days of telemetry, while others have been developed in the Programmable Logic Controller (PLC) industry. TEST has developed its own protocol which is designed to handle the functions actually required for this type of equipment in typical remote SCADA applications. Admittedly, this protocol is quite a bit different from the traditional protocols used by other developers. But these differences are what makes TEST's systems so practical and easy to use.

All protocols do basically the same thing. They allow information to be reliably sent between two or more devices. The exact way that each protocol performs this function is as varied as the equipment on which they are implemented. Generally speaking, each protocol evolved around a particular type of equipment or a specific application. Therefore, each protocol has advantages in certain applications but may not be as appropriate under different circumstances.

RTU talking to host with 1 and 0
from Open Arch Document

12.2 TEST SCADA PROTOCOL (TSP)

TEST SCADA PROTOCOL (TSP) is TEST's answer to the problems of typical oilfield and industrial SCADA. TSP was developed to serve a high technology Personal Computer based SCADA system first implemented in 1987. It has since developed into a complex language of its own that is ideally suited to modern PC based SCADA solutions. TSP has built-in support for the many SCADA features designed into TEST systems. It also supports any type of communication media including hard wire, microwave, cell phone, radio, and fiber optics.

TSP is a command based language that offers about one hundred different data transfer and command functions. Many systems only need a handful of the options, while others make use of almost all of the capability. There is only a single version of the program that is updated constantly, and the protocol is constantly extended to include new features. This is considerably different from other RTU and PLC protocols

that are very inflexible and cater to only a specific application. TSP is more like a programming language that can be reshaped with its own built-in resources to suit each individual application. Therefore, although all installations are similar they each possess unique functions that are provided by the TSP language.

TSP works with command and response lines that are always simple ASCII text lines. The command processor accepts these lines, executes a function based on the contents of the line, and often generates a response line. This response line may actually serve as a command line to another unit. In this way, various copies of the software communicate by sending text commands to one another. These lines can be easily viewed by an operator or technician so that all activity of the system can be monitored and diagnosed.

This is a very simple concept, one that is difficult for veteran Programmable Logic programmers to understand. Older protocols were based on complex command and response codes, often involving only HEX or Binary coded numbers. While these are convenient for the hardware, they are impossible for a person to understand without a lot of assistance from either a computer or a special piece of programming hardware. It is not uncommon to hear of programmers taking a week to connect two systems located five feet from one another. A large part of this is due to that fact that the programmers cannot easily deal with the computer-to-computer transmissions. The heart of TSP is that it is just simple TEXT messages that make sense to both the computers *and* the people who use them. Although this places a much higher burden on the computer, it makes the user's job much easier. And this reduces the most expensive component in the system: manpower.

So, we agree that TSP uses simple TEXT and that it is easy to read. So what can we do with it? Well, imagine yourself sitting with a blank piece of paper and getting to write down all the things you wished your SCADA system could do. That is what TEST did (and is still doing), and the results are found in TSP. There are commands related to data transfer, calculations, I/O configuration, communications control, system setup, alarm functions, and numerous other functions. Each command line starts with a simple keyword such as SCAN or DATA. The rest of the line provides information to or from a built-in function. In essence, the software reads these text instructions and returns results in simple words and numbers. No computer voodoo, no magic. It is so simple, it is understandable that someone who has spent years dealing with older PLC and RTU systems would be skeptical of TSP's power.

One immediate reaction from new users is "Hey, I can't type! I can never use this thing!" Well, we agree that most field users cannot type and should not be expected to do so on a routine basis. So, all of the TSP functions can be easily automated through the use of simple TEXT files (called command or RTU files) that permanently store the often used functions. These files are activated by numerous mechanisms such as a menu selection, alarm condition, communications requirement, time of day, and several more technical matters. But the end result that is no matter how complex the operation, the software is simply "reading" some text. The text can come from the keyboard, a DOS file, a command library, a built-in message system, or from another unit. All of these methods have their particular application, but they all work the same. So, if you can do something by hand at the keyboard, you can probably do it to a remote system by having one copy of the software send the messages to another copy over a phone or radio.

That is the essence of TSP. It is simple for the operator to use and understand yet provides unlimited capability through the power of the PC. There is no other protocol that even comes close to providing the level of features that now exist in TSP, and the list of features will continue to grow.

12.3 TSP PROTOCOL MECHANICS

TSP sends and receives commands in text mode in one of two basic formats. When working at a local CRT and keyboard TSP uses the "Human" or TERMINAL mode. This is a simple text protocol where lines are entered with the ENTER key at the end of the line. This generates a Carriage Return that tells the program that the line is complete and ready for processing. Prior to hitting ENTER, the user can backspace and make corrections to the line because the program does not process the line until it is complete and the ENTER key is pressed. This is fine for manual operations because a person is there watching for mistakes and making sure that the system responds properly to each command.

Computer-to-computer communications (RTU Mode) uses the same text messages as manual mode but packages the messages with special codes. This packaging puts the messages into an envelope to control sequencing and detect errors in transmission often found in industrial applications. The intent is to allow TSP commands to be sent among the units in a reliable and coordinated manner. The packaging is done automatically by the system during data transfers, and is never used manually.

TSP is not like other RTU protocols that are limited to a few select commands that request simple binary data transfers. TSP allows any command that can be entered at the keyboard to be sent to another unit with complete security. So, TSP is actually the language of the SCADA system as well as the method of communications all rolled into a single concept.

When a task is in RTU mode, it can be switched to TERMINAL mode to make it easier for a person to operate. This is done with the special key sequence consisting of a Control-A followed immediately by the characters *TERMINAL. The Control-A starts the RTU mode line and the *TERMINAL is a special escape password that takes a task from RTU mode into TERMINAL mode. From that point, commands can be entered without concern for the normal packaging required for remote communications. If a task times out due to a lack of command, then it will automatically return to RTU mode to await further computer-to-computer style communications.

The structure of a TSP message packet (in RTU mode) is as follows:

SOH	ASCII Start of header, Control-A
MD Dest/	Multi Drop Destination ID Name (optional)
MD Source/	Multi Drop Source ID name (Optional)
Cmd No.	The command sequence number of the sending unit
ACK	ASCII ACK to indicate that previous message was OK
ACK No.	Message number being ACK'd by this response
SOT	ASCII Start of Text code, Control-B
MESSAGE	Any TSP message in variable length plain text
ETX	ASCII End of Text, Control-C
CHECKSUM	Numeric sum of all characters after the SOH
CR	ASCII Carriage Return, Control-M

This is a fairly complex protocol that is designed to allow efficient processing in poor industrial communications systems. In RTU Mode, every message sent by a system must receive an acknowledgment (ACK) from the other unit to indicate successful reception and processing. Rather than using an entire transmission for a simple ACK, the protocol allows a response to carry an ACK as well as another message back to the sending unit. This cuts the transmissions in half which greatly increases the speed of packet based communications systems. This complexity is handled by the program itself and should only be visible during debugging sessions. However, it does no harm to be aware of what is going on behind the scenes as TSP transfers data in the background.

All lines in RTU mode must begin with an ASCII START OF HEADER code SOH, which is CONTROL-A. In RTU mode, nothing is processed until the SOH is received. This is why the CONTROL-A must be entered when switching from RTU mode to TERMINAL mode as described above. Any characters received in the serial port prior to the SOH are simply discarded.

After the SOH is received, the line is buffered by the comm port driver until a carriage return CR is received. If a SOH is received prior to a CR, the line is simply restarted. After the CR terminates the line, the line processor uses the checksum to determine if any problems occurred in the reception. This is quite possible in typical SCADA systems due to problems with the phone or radio system. If the line checksum calculates properly, the line is accepted for processing. If not, then the system sends an ASCII NAK (negative Ack) to the other unit and discards the entire line. *Note: Multi-drop systems do not send NAKs because they cannot be certain that the garbled message was intended for them.*

What happens after the line is accepted depends on the Multi-Drop nature of the link. In multi-drop mode, the system will check the Destination ID to see if the message is intended for this unit. If not, then the

line is simply discarded. If it is, then the source and destination IDs are pulled out of the line and stored for use during the response phase of processing. If this is not a multi-drop, then the Source and Destination IDs will not be in the message envelope.

If the line is received properly, then the incoming command number is checked to see if it is higher than any previously processed lines. This prevents duplicate lines from being processed more than once. If it is a duplicate, the RTU simply sends an ACK because the proper response should already be on its way to the other unit. If it is not a duplicate, then the line is passed further on in the processing system. *Exception: An incoming command number of 1 is always acceptable and resets the command sequence counter on the receiving unit. This is called resyncing and is useful for continuous connections that never disconnect.*

If the incoming command number is 0, then the receiver assumes that the command does not ever need an ACK. It will process the line but will not send an ACK to the other unit after processing is complete. This is often used by the system to respond to simple commands that do not generate a data reply. The unnumbered reply will indicate that this is simply an OK but don't bother to send me an ACK to my ACK.

If the line was properly received and the sequence number is high enough (or equal to 1), the line is picked apart into separate sections. This is referred to as "parsing" the line. The ACK number is checked to see if it is the number we expected from a previous transmission. For example, if one unit sends out command number 10 it will expect an ACK for this command on the next line that comes in. If the ACK does not come in, then the program will retransmit the previous line again with the assumption that it got lost on the way to the other unit.

After the line is parsed, the text portion of the message is passed along to the command processor just as if the line had been entered locally. The command processor does not get involved in the details of the TSP envelope. Therefore, commands that enter the processor from the command line, a text file, an inter-task message, or a network connection are all processed identically. The resulting output from the command will be rerouted back to the sender automatically by the communications processor. With this system, command processing is generic regardless of the source of the command line.

12.4 TSP DOWNLOAD PROCESS

One of the primary purposes of the SCADA system is to transfer data from one system to another. Each system has unique communications and data channel requirements, and the program allows for each installation to be set up to suit the needs of the user.

As in many SCADA systems, the basic data transfer operation is done with SCAN commands and their associated DATA command responses. Selecting how and where the commands originate depends on the logistics of the system. The primary consideration is the speed of the communications link. In a fast system, it is easy to have the "HOST" system send SCAN commands one at a time and receive DATA responses one at a time from the RTU. All the SCAN commands necessary to get the desired data from the RTU can be processed from a command file located at the Host.

In a slower half-duplex radio system, however, each SCAN command sent by the Host will take a considerable amount of time to be sent by the Host and then acknowledged by the RTU before processing. In these cases, it may be better to have a prepared file at the RTU containing the SCAN command lines. In this case, the Host sends one command to the RTU telling it to read the file, rather than sending each individual SCAN line to the RTU.

The advantage of the Host controlled method is that any changes needed in the process can be easily made at the Host computer. In the RTU controlled method, the prepared file stored at the RTU must be altered at the remote site. No matter how the sequence is done, the same processes are used to send and receive the data.

When a connection is made between two units the CONNECT files at each unit will be processed if they exist. The unit that initiated the communications (for whatever reason) will then process a LINK file if it exists. If

the LINK file does not exist, the computer will process its own default commands depending on whether the unit that initiated the communications is a Host or RTU. The default command for a Host unit is BLOCK READ DOWNLOAD. The default command for an RTU is READ DOWNLOAD. The DOWNLOAD file contains the SCAN commands necessary to transfer the RTU's data to another unit. Because of the default commands, the more simpler systems do not require a LINK file.

The LINK files described in Section 8.7 are set up for each of the defined communications links. The actual names of the files are LINK1, LINK2, etc, with one for each link. Each link represents a separate phone number (or radio call sign), and is therefore dedicated to a particular RTU or Host. So, each LINK file can be tailored to do the steps necessary for each RTU. For more information about CONNECT files and LINK files refer to the COMMAND FILES section of this manual.

12.5 TSP DATA SCAN FORMATS

The RTU program uses a combination of SCAN and DATA commands to transmit data from one unit to another. The SCAN command is used to request information and the response is a DATA command. The SCAN command specifies a range of channels and the DATA command is returned with the same range of channels followed by their corresponding values. The values are listed one after another on the line in ascending channel order. The SCAN command can also use code letters after the channel range to specify the format of the values to be returned in the DATA command. For example, consider the following SCAN line:

SCAN S1:S8 E

This line requests data for status channels 1 through 8 in engineering units. A possible response might be:

DATA RTU1.S1:S8,E,0,1,0,0,1,0,1,0

Note that the DATA response always includes the RTU from which the data originated. The channel range will be the same as in the SCAN command that generated the DATA response, except that out of range channels will not be included. For example, if an RTU has only 16 status channels, a SCAN S1:S32 request will be modified in the DATA response to S1:S16, and only 16 channels will be included.

If no code letters are given in the SCAN command the letter "E" is assumed. The "E" indicates that values are to be returned in engineering units which represent the current value of each channel. There are several codes available which can be used to get various information from another unit and in various formats. Some codes can be used in conjunction with other codes and not all codes are applicable to all channel types. For a complete list of available codes refer to the SCAN command in the TEST SCADA PROTOCOL - COMMAND REFERENCE MANUAL.

After processing the SCAN command, the DATA command will be sent back with the same codes that were specified in the SCAN command. The data and format of the data returned depends on how the unit that processes the SCAN command interprets the code letters. Any conflicting or invalid codes will be ignored.

Normally, the Engineering units option will be used to have the RTU send down calculated values for each channel. The precision of the transmitted data for each channel is determined by the Decimal Places parameter that is unique for each channel. This parameter can be set for a channel by using the CONFIG or PROGRAM command. If this parameter is negative for a channel, the precision of the transmitted value is determined by a global variable. The SET PLACES xx command is used to determine the precision of values for all channels that have a negative value for the Decimal Places parameter. The only variation regarding precision is that any value exactly equal to 0 will be represented by a single 0. This will save space on the line when many channels have a zero value.

If the Raw option is used, the appropriate internal value for each channel will be transmitted. For analogs this is the 16 bit signed integer that is obtained from the A/D board. For counters this is the raw pulse count. In the case of Status and Output channels, the Raw format packs 8 channels into a single number. This allows for

a much more compact transmission of digital channel types. For example, the line

DATA S1:s8,E,0,1,0,1,1,1,0,1

represents the same data as the line

DATA S1:S8,R,93.

The first line uses Engineering units and the second uses the Raw format. They represent the same data because 93 decimal is the same as 01011101 binary, but the 93 is much more compact. This is especially useful when there are several active data points. For example, in Raw format a single 0 can be sent instead of 0,0,0,0,0,0,0,0 in the Engineering units format.

12.6 MULTIPLE TSP CHANNEL SCAN FORMATS

Multiple channel types can be transferred on a single line. This is useful in half-duplex systems by allowing more data to be sent in each packet. The format is very similar to the normal single channel type SCAN line. The additional channel ranges and code letters follow the first one on the line. *All channels in a single SCAN line must belong to the same logical RTU* because only the first range of channels in the resulting DATA line will contain an RTU ID. For example, the line

SCAN S1:S8 R O1:O8 R A1:A4 E

may produce the following response:

DATA RTU1.S1:S8,R,22,/O1:O8,R,13,/A1:A4,E,123.4,567.8,12.3,45.9

Note that each additional channel range on the resulting DATA line begins with the slash character. The originating RTU designation is only provided on the first channel range (RTU1 in this example) and all subsequent ranges on the line are assumed to be for the same RTU.

Multiple scans must be set up so that the resulting line will not exceed 200 characters. This is the maximum line size allowed in all transmissions. In environments where communications are poor, shorter lines may produce better overall results due to the difficulty in sending long transmissions.

12.7 SENDING TSP DATA TO A FILE

The SCAN command can also be used to send data to a file rather than to another unit. The format of the data sent to a file is exactly the same as the format of the data sent to other units when a SCAN command is processed. The file would contain DATA commands and could be used at a later time to load the stored information on to any system running the RTU program.

A simple RTU command file could be used to send the responses from some SCAN commands to a file. The sequence is to open an output file, process the SCAN commands, and then close the output file. For example, consider this RTU command file:

```
file open wc458.dmp      ; open a "dump" file
scan s1:s32 r
scan a1:a6 e
file close               ; close the "dump" file
```

This file will perform the steps necessary to get a dump of status and analog channels into an output file called WC458.DMP. The contents of the file might look like this:

DATA WC458.S1:S32,R,12,15,0,3

DATA WC458.A1:A6,E,12.5,1433.6,34.3,42.7,12.65,8.34

This file could then be read at a later time to load in values for status input channels 1-32 and analog channels 1-6.

When dumping a large number of channels it is best to break them into groups that will keep the line lengths less than 100 or so characters. This simplifies data transmission as well as editing with word processors and text editors. For example, a system with 32 analog channels may use the following scheme:

SCAN a1:a8 E
SCAN a9:a16 E
SCAN a17:a24 E
SCAN a25:a32 E

For more information about sending data directly to a file refer to the FILE command in the TEST SCADA PROTOCOL COMMAND REFERENCE MANUAL.

12.8 EXAMPLE TSP DOWNLOAD FILES

Download files consist of all commands necessary to transfer data between two units once a connection has been made. A typical download file consists of the following steps:

1 - Slight wait to allow modems to sync.
2 - SET ONLINE ON to prohibit processing on disconnect.
3 - SCAN lines to send appropriate data.
4 - LINK RESETS to clear RTU and/or Host call outs.
5 - BYE command to terminate.

Consider the following example DOWNLOAD.RTU command file located on an RTU:

```
; DOWNLOAD file for RTU at MP36
sleep 2                 ; allow the modems to lock in
sele mp36               ; select proper RTU designation at the RTU itself
block sele mp36         ; tell the Host to switch to MP36
scan a1:a16 e s1:s8 R o1:o8 R
scan c1:c3 E v1:v8 E@   ; finish data transfer with time stamp
link reset                    ; reset the call out request locally
block link reset              ; reset the Host's call out request
set online off          ; on-line state no longer required to go on
block BYE                     ; tell Host to terminate
bye                     ; terminate at the RTU
```

This example file contains the steps needed for a single RTU download to a Host. Other systems may require more scan lines, more RTU selections, etc, but the steps are similar. Note that this file is stored and processed at the RTU, not the Host, because all scan lines are processed locally. For a Host controlled download file the process would be similar except that the SCAN lines would be replaced with BLOCK SCAN lines. The BLOCK command tells the Host to send the line to the RTU for processing. Without the BLOCK keyword, the SCAN line would be processed by the Host rather than the RTU, resulting in an UPLOAD from the Host to the RTU. This is a common error, so make sure the difference is clear.

Sometimes it is desirable to send data from the Host to the RTU. This is often done with VALUE channels that are modified at the Host. The command file to do this would be stored and processed at the Host system and probably activated by a menu selection. Note that this file would be processed by one of the communication tasks, not the local keyboard task. Because it is often desirable to monitor the activity of these procedures, a number of MSG lines could be placed in the file. This would allow the local task to monitor the progress of the procedure. For example, a file that sends VALUE channels from a Host to an RTU could be as

follows:

```
MSG CALLING MP36 to set new value channels
sele MP36               ; select rtu at the Host
dial              ;call it using the default phone number
msg Waiting for a connection
wait 30 conn      ; wait up to 30 seconds for a connection
set online on     ; quit if we did not connect
msg RTU is online. Sending Values.
scan v1:v8 E      ; send from Host to the RTU
msg Values sent. Saving data at the RTU
block save
set online off
msg Procedure complete. Terminating call.
block BYE               ; tell the RTU to terminate.
bye               ; terminate locally
```

The MSG lines are just information displays at the main console and do nothing to the actual process. In order for the main console task to see the messages it must be at the command line prompt. If not, the messages will be stored in a queue until the task is idle. To insure that the CRT task is idle you can insert the following at the start of the command file:

STUFF 0 CAN ESC CAN ESC

This will force the ESCAPE and CONTROL-C characters into the keyboard buffer of task 0, the local CRT task. If the task was displaying any channel information, it will exit back to the command prompt so that the messages can be seen. However, if task 0 is editing or configuring a channel, the resulting confusing may not be desirable. Therefore, use this procedure with caution and only in files that are started by MENU picks. At these times, it is unlikely that task 0 will be doing any editing or other special screen oriented functions.

12.9 OPEN ARCHITECTURE CONSIDERATIONS

This section details how TEST's SCADA PROTOCOL (TSP) can be used to exploit the advantages of what has come to be known as *OPEN ARCHITECTURE.* It explains how TEST systems communicate with each other and with the "outside world" in a manner that makes the best use of available hardware and software resources. This document is not an in depth technical description of TSP or any other protocol. It is intended for technical readers, especially those considering a substantial SCADA system that must satisfy a wide range of end user needs.

[open arch oval]

The term *Open Architecture* has begun to appear often in SCADA system documents and specifications. Depending on the context of the statement, the term can have slightly different meanings and consequences. But the general intent is to describe a SCADA system designed to easily communicate with other types of systems, including those from different vendors. Unfortunately, this concept is much easier to specify than it is to implement. Contrary to what is often stated in the press, there are no clearly defined communication standards that allow instant connection of otherwise incompatible systems. Likewise, the theoretical interfaces often cited in specifications fail to recognize that so called protocol "standards" are often more like "suggestions."

So the seemingly simple term Open Architecture does not refer to a standard option available on any piece of SCADA equipment. It more correctly refers to an overall system design concept where various parts of the system are allowed to do what they do best, while maintaining a minimum level of communication between components that is based on some common theme.

12.10 MODBUS AND OPEN ARCHITECTURE

Yes, we admit that TSP is incompatible with other systems. But we can also say *those other systems are incompatible with ours*, and such is the nature of the Open Architecture dilemma. Although there is no true standard with which to be compatible, one popular protocol has emerged as a defacto standard for connection of diverse systems. Like many other vendors, TEST has chosen the Modicon MODBUS PLC protocol as its Open Architecture interface. TEST's implementation is a very complete and comprehensive MODBUS protocol that allows outside systems to make best use of TEST's SCADA capabilities.

MODBUS is a very simple, low level protocol originally designed to interface closely coupled PLC equipment on hard wired communications links. The mechanism allows a master unit to send and receive binary data with slave units on a single link. Although this is a primitive system rooted in 1960's technology, it is well understood and available on a wide variety of equipment. Its emergence as an Open Architecture standard attests to the simplicity of MODBUS rather than its actual capability. In short, MODBUS forms the lowest common denominator that can be shared with all systems on a single network.

12.11 DIVIDING SYSTEM FUNCTIONS

A good question regarding open architecture design is which functions are best performed by which pieces of equipment? Why should several different protocols be used in one system? What are the advantages of mixing and matching RTU, PLC, and Computer devices? These are good questions, and the answers will depend on the needs of each installation. However, there are several common factors that can be considered on almost any system.

A typical large scale SCADA system can be divided into three distinct areas. DCS systems have several more subdivisions, but these can be ignored for purposes of this discussion. The three main areas for SCADA are:

- Remote Terminal Units (RTU)
- SCADA Concentrator (Master Terminal Unit, or HOST computer)
- Management Information System (MIS)

The RTU is the processing hardware and associated end devices located at the remote location to be monitored or controlled. A single system may have varied requirements for the RTUs, but they will usually fall into a range that can be met by two or three similar RTU devices.

The SCADA Concentrator, Master Terminal Unit (MTU), or Host computer is responsible for coordinating the activities of the RTUs in a real-time environment. A properly designed Host will handle all communications,

real-time alarm display and logging, and diagnostic functions directly related to the monitoring and control functions.

The Management Information System (MIS) provides historical data storage and possibly network data distribution. This may include sophisticated operator interface software that offers enhanced graphical display and database functions.

The needs of each portion of the overall system can best be met with different types of equipment and software implementations. Trying to make them all conform to a single low level protocol will likely cripple the benefits that made the equipment a good choice in the first place. A more sensible approach is to let each portion be served by the best equipment and then connect them as required with an Open Architecture protocol such as Modbus.

12.12 SCADAWARE AND OPEN ARCHITECTURE SYSTEMS

Although TEST's SCADAWARE program can provide all necessary SCADA functions, some installations desire or need capabilities not present in the standard program. These systems can still benefit from TEST's system by dividing the system into appropriate sectors and letting the best equipment be installed for each sector.

TEST has performed several Open Architecture designs that connect its own SCADA systems with DCS and Operator Interface systems from other vendors. These systems allowed TEST's RTU equipment and SCADA Concentrator computers to do what they do best while providing inter-system compatibility with other systems. The result was a reliable, cost effective SCADA system that performs all the real-time processing requirements with a minimum of complexity. The external systems interface to TEST's systems with minimum data transfers to coordinate overall system activity or provide elaborate operator interfaces.

Several advantages obtained by using TEST's system as a "front end" coordinator to a complex Open Architecture SCADA system are:

1. Proven world class PC based RTU and Host Computer technology.

2. Easy to configure and use SCADAWARE software.

3. Fast-Track system implementation for key system components - measured in hours, *not months or years*.

4. Higher through-put of real-life SCADA information on typical radio and phone based communications links.

5. Low-cost industry standard PC component designs for all parts of TEST's PC Based Smart SCADA Systems.

7. Cost Effective SCADA Node RTU's for low point count locations.

8. Minimal system development and installation costs.

9. Non-obsolescence and tremendous expandability through TEST's use of commonly available industrial PC based equipment.

12.13 DCS INTERFACE EXAMPLE

A large DCS/SCADA system was designed, fabricated, and installed by TEST for TARIM Development, part of the China National Oil Company. The system uses 23 TEST RTU units located at field well locations remote to the main facility. These RTU's communicate by radio to a central PC acting as the SCADA

concentrator. All of TEST's advantages and benefits are used by this stand-alone system to solve the telemetry and remote control needs of the system.

The TEST SCADA concentrator communicates over MODBUS to a Fischer & Porter System Six DCS network that controls the oil and gas processing facility. The DCS does what it does best - complex facility and process control. TEST's system does what it does best - real-time SCADA. The marriage of the two systems is very simple and allows each to operate optimally.

An unforeseen advantage is the ability to use TEST's SCADA Concentrator as a gas flow computer by having the DCS download real-time transmitter values for calculation by SCADAWARE. This method was chosen because TEST's AGA-3 Gas Flow calculations were superior and easier to implement than the equivalent function on the F&P DCS. The net result is the seamless integration of two diverse systems into a single hard-working network that makes the best use of the most valuable resource: *the budget.*

dcs example

12.14 OPERATOR INTERFACE EXAMPLE

Although TEST's SCADAWARE has adequate database and graphics capabilities, some installations desire a more comprehensive operator interface. TEST's Open Architecture interface allows its systems to be used with external programs in much the same way as a normal RTU or PLC based system. Programs such *as ICOM WinView, WonderWare Intouch, or Intellution The Fix* are interfaced to a TEST System just as they would to a standard PLC. Options such as networking among Personal Computers or high performance workstations are commonplace with these packages, and their use provides a convenient bridge between TEST's system and these complex MIS applications.

Employing a TEST system allows most of the routine work normally associated with SCADA to be off-loaded to the TEST Host PC. Functions such as radio key control, PID loops, AGA-3 gas flow calculations, alternate comm path selection, and instant operator interfaces are very easy to implement in TEST's SCADAWARE.

[operator interface example]

The powerful graphical interface and database capabilities of the external software are not burdened with the functions so easily handled by SCADAWARE. This makes the combined system far more practical because the sophistication of TEST's SCADA is used to provide a reliable system that works under the worst of real-life conditions. This frees the external operator interface to do its job without cumbersome programming of chores routinely done by SCADAWARE. Again, the division of labor between TEST's system and the external program provides the most productive, cost effective solution to a high performance operator interface situation.

Note: The external program does not have to be PC based. Any program that uses the MODBUS interface can easily be linked to the TEST system. This includes existing mini-computer and main-frame applications that require expansion with more modern RTU's such as TEST's Smart and SCADA Node units.

12.15 TSP ADVANTAGES OVER MODBUS

Why is TSP a better choice than MODBUS for typical oilfield and industrial SCADA applications? The answer to this question lies in the roots of the two protocols. MODBUS was originally developed to solve PLC communications problems. Programmable Logic Controllers (PLCs) were designed for local, high speed control in industrial situations. *Period.* They were not designed for remote operations, although many of them provide some facility to send and receive data scans over a dedicated communications link.

PLC protocols also have no built-in ability to perform typical SCADA actions such as dial a phone, key a radio, or perform an AGA-3 gas flow calculation. The PLC protocols are designed for high speed, local, binary data transfer. Data transfer covers about 5% of what a SCADA system must do, and PLC systems have no provision to perform the remaining 95% of the job.

On the other hand, TSP has its roots in modern PC based SCADA. It is a command file driven system that uses easy to understand ASCII Text messages instead of the cryptic binary or hex digital codes used on a PLC.

```
TSP example:
      Host sends  "SCAN A1:A8 E" (request analogs 1 to 8 engineering
units)
      RTU Responds      "DATA A1:A8, E, 123.1, 2.3, 3210.65,..."(simple
numeric responses)

MODBUS example:
      HOST sends "02C3B34572A8D2B3" (cryptic Modbus request)
      RTU Responds "03D4243CD80942345198B37F" (even more cryptic response)
```

Which would you rather program or troubleshoot? Unlike PLC protocols that only understand one's and zero's, TSP understands SCADA at all levels and can process multiple data types in a single data burst. It is easy to send status, analog, counter, and other data in a single transmission. This saves time and battery power in radio based systems that place demands on battery power with every transfer. Another important feature is that no special equipment is needed to diagnose or monitor all actions of a TSP based system, *including those on the Modbus Open Architecture* link. Installations all over the world have shown that TSP is the most efficient method of sending typical RTU data in real life situations.

TSP is also considerably more advanced than other RTU protocols because it is actually a programming language in itself. The basic functions of scan and control have been greatly enhanced to provide over 100 different commands and functions within the simple command language. Unlike Modbus, TSP is a peer-to-peer networking language whereby any device can send and receive from any other. This greatly increases the flexibility of new installations and insures that future changes can be handled with minimal impact on the overall system design.

12.16 TEST'S MODBUS INTERFACE

MODBUS defines only low level data types such as bits and bytes. TSP provides high-level data types such as Analog Values and AGA-3 gas flow meter calculations. The interface provided by TEST allows all of the important TSP data types to be easily "mapped" into equivalent low-level Modbus coils, inputs, and registers for transport to external systems.

MODBUS DATA TYPES: Documented: Bits, Words. Undocumented: IEEE Floating Point. No specific purpose assigned to any data value.

TSP DATA TYPES: Bits, Words, and IEEE Floating Points for Status Inputs & Outputs, Analog, Value, PID, AGA-3, Timer, Counter, Special Function, Totalizer, Low Alarm, Hi Alarm, Alarm Delay, Deadband, Alarm Times, Counter/Totalizer Reset Time, Current Alarm mode, and many others..

Built-in TSP Features such as alarm status, deadbands, setpoints, and other internal system variables are easily accessed on the Modbus link. The relatively low point count limitations of Modbus are overcome by letting a single TSP/Modbus interface simulate up to 16 separate Modbus RTU addresses. This allows thousands of equivalent Modbus values to be transferred efficiently and quickly using standard Modbus data transfer commands.

Please refer to document 1220, MODBUS Open Architecture Manual, for more information and

examples on TEST's extensive Modbus implementation.

13 DATA LOG SYSTEM

13.1 LOGGING CONCEPTS

The RTU program supports a log system which is used to record channel values and text messages in files called "log" files. These files are standard DOS files that can be edited with any text editor that can handle simple ASCII file formats. The files are normally named after the logical RTUs that generate the information and have a DOS file extension of .LOG. For example, the log file for WC458A is normally called WC458A.LOG. This default file name can be changed by using the LOG FILE command. Each logical RTU has a separate file even on Host systems that support many individual RTU locations.

There are 3 ways in which data can be logged. First, a channel's value can be logged automatically whenever it enters an abnormal state. This type of log is known as an Abnormal log. Second, a channel's value can be logged automatically whenever the channel is reset. Third, a channel's value or a text message can be logged at any time by using the LOG command either manually or within a command file. Using the LOG command to log a channel's value is known as a Data log. Using the LOG command to log a text message is know as a Message log. Each of these types of logs is explained in more detail in the sections that follow.

Note that the Data Logging described in this section is not the same as SCADAWARE's Database system. The Database function, described in a separate document, stores data at fixed intervals in a ridgid file structure that allows random access based on time and date. The logger is designed for periodic storage of alarm and status information that will be accessed in sequential form.

13.2 ABNORMAL LOGS

Abnormal logs are used to record the time, date, and value of a channel when a channel enters an abnormal state. Each channel on a system can be individually configured to generate abnormal logs by using the CONFIG or PROGRAM command. A channel does not have to be setup as an alarm type channel to generate an abnormal log.

The process of generating abnormal logs at a Host unit is a little different from the process that takes place at a RTU. When an abnormal condition occurs at a RTU for a channel that is setup to log abnormal conditions, a log is immediately performed using the current time and date. However, at the Host the log time should not reflect the time that the abnormal condition occurs locally, but rather the time that the abnormal condition occurred at the RTU. Therefore, the logging process at the Host is *delayed* to allow the correct time and date of the abnormal condition to be obtained from the RTU. To fully understand how the log process works for abnormal logs, an explanation of where the log times and dates come from is necessary.

All channels, whether at a Host or a RTU, have an alarm time and date associated with them. These times and dates are used to keep track of when channels make the transition from a normal state to an abnormal state. By default, these variables are set to 00:00:00 and 01/01/00 for all channels. The first time a channel encounters an abnormal condition these variables are set using the current time and date. For an alarm type channel, this time and date can not automatically be set again until the alarm condition has cleared, the alarm has been acknowledged, and the channel reset. For other channels, this time and date are eligible for another update as soon as the abnormal condition has cleared. When an abnormal condition occurs again, this time and date will be updated again using the current time and date.

When an abnormal condition occurs at a RTU for a channel that is setup to log abnormal conditions, the alarm time and date for that channel will be set using the current time and date. The logging process will then immediately perform the log using the alarm time and date. This is a very simple process and is all that is required to log abnormal conditions for local channels.

At the Host unit, the values of remote channels are received over a communications line in the form of DATA commands. When an abnormal value is received for a channel that is setup to log abnormal conditions,

the alarm time and date for that channel will be set using the current time and date. However, the logging process is not immediately performed. Following the update of a channel's value should be an update for the channel's alarm time and date (using the A option in the SCAN command). This will override the alarm time and date just set for that channel with the alarm time and date from the RTU. This will reflect the actual time that the abnormal condition occurred. The last DATA line of the download process usually contains either the @ or @ENDS option. It is at the time that one of these options gets processed at the Host that the logging process takes place.

13.3 ALARM RESET LOGGING

The system can automatically log the return-to-normal actions of any channel. The LOG ABNORMAL? setting can be made to affect both the New Alarm and Reset steps in the alarm sequence. The default condition is to log only new alarms. If the command

```
SET RLOG ON     ; turn on the Reset Log function
```

is entered the logging system will also log channels at the time they are reset. Channels still have to have the LOG ABNORMAL? setting turned on in order to be considered for logging.

Note that the RESET log occurs when the channel is reset, not when it becomes available for reset. Points set to AUTO RESET AFTER ACK will automatically return to normal when the channel goes within tolerance. Those channels will be Reset Logged at that time. Points with AUTO RESET AFTER ACK set to false require a manual or programmed RESET function to occur. Those points will be Reset Logged at the time the reset actually occurs, not when the point clears and becomes eligible for reset.

13.4 DATA LOGS

Data logs are used to record the time, date, and value of a channel at any given time. A data log can be performed at any time for any channel by using the LOG command. This command can be processed from the keyboard or from a command file. The agenda system can be setup to log a channel's value at a specific time of day without regard to the state of the channel.

Data logs always use the current time and date for the time stamp of the log record. This is different from abnormal logs which use the alarm time for the individual channel.

```
LOG A1:A5               ; Store first 5 analogs current values
```

13.5 MESSAGE LOGS

A message log is simply a text message that is sent to the log file with a time and date stamp. A message log is performed using the LOG MSG command. Message logs and data logs can be mixed as needed in the same log file.

```
LOG MSG This message will be stored in the log
```

13.6 FIRST OUT ALARM LOGS

The log table for each RTU reserves entry number 0 for a special type of log called the first out alarm log. The first out alarm log keeps track of the first channel to go into alarm for an RTU. Once a first out alarm is recorded, no other alarm can become the first out until the current one is acknowledged. Even after being acknowledged, the last first out alarm log will remain until the next first out alarm occurs and replaces it.

In order for a channel to be logged as the first out alarm, it must be setup as an alarm type channel. However, it does not have to be setup to log abnormal conditions. The current first out alarm for an RTU can be examined at any time by using the LOG FIRST command. The first out alarm can also be downloaded to a Host unit by using the SCAN FIRST command. The first out alarm does not get saved to disk like the other types of logs.

13.7 LOG SEQUENCE

All logs are written to a temporary RAM based log list at the time the log occurs. The logs are subsequently written out to a disk based log file at some regular time interval as specified by the LOG EVERY xx command. The temporary memory log allows for a fast response to rapidly occurring abnormal conditions because the system does not have to wait for relatively slow disk I/O to occur for each log entry.

By default, the UTILITY task is responsible for transferring the log data from memory to the disk based log file. This can be changed to another task by using the LOG TASK xx command to specify another task to receive the messages. The use of a separate task for writing to disk provides a way for this lower priority function to be processed as processor time becomes available. The system provides a memory buffer sized to handle 16 instantaneous log entries from each RTU. This size can be changed with a LOG xx entry in the DAT file as each RTU is defined. This value would control how many *concurrent* memory logs can be in the buffer waiting to be written to disk. Once written to disk, however, the size of the log list is limited only by available disk storage.

13.8 LOG FILE FORMAT

Each entry in a log file consists of a number of text fields separated by commas. This is referred to as a "comma delimited" file in DOS terms. Any program that can read (or "import") a comma delimited file should be able to at least input the raw information from the log file.

The log files store data in a compressed format where the minimum amount of information necessary to identify the RTU, channel, and log data is used. This reduces the size of the files to a minimum.

The fields of the log file are as follows:

1. Full RTU and channel name in "dot notation" or a #0
2. Log type (A, L, M,or R)
3. Time
4. Date
5. Value or Message

Field 1 represents a full RTU and channel name. For abnormal logs and data logs, the RTU and channel name are provided in the standard dot notation used throughout the RTU/SCADA software system. This is a means of identifying the exact point that originated the log entry. The entry consists of two parts separated by a period (or "dot"). For example, the entry VR167A.S3 indicates the data originated at an RTU named VR167A and the point was the third status point. In the case of the message logs the channel will always be shown as #0, indicating no channel type or RTU.

Field 2 is a single letter that indicates the type of log. The code letters used are:

A	-	Abnormal Log
L	-	Data Log
R	-	Return to Normal Log
M	-	Message Log

Fields 3 and 4 are the time and date that the log occurred. Time is provided in 24 hour format as HH:MM:SS. The format of the date will depend on the country in which the system is operating. The following commands can be used in the config.sys file to cause the dates to appear as follows:

```
COUNTRY = 001  <-- United States  mm/dd/yy (dos default)
COUNTRY = 003  <-- Latin America  dd/mm/yy
```

Field 5 is either the value of the channel at the time the log occurred or the text of a message log. The value for status type channels will always be either 1 or 0, while analog type channels will have their values in a floating point format like 12.4.

Consider the following example from a log file:

```
WC457AJ.A1,L,13:42:24,08/07/89,112.4
WC457AJ.A2,L,13:42:24,08/07/89,2436.6
#0,M,13:43:03,08/07/89,Communications Failure
WC457AJ.S1,A,13:47:22,08/07/89,1
```

These logs come from an RTU named WC457AJ. The first two logs occurred at 13:42:24 (i.e. 1:42 p.m.) on Aug 7, 1989. The values at the time of the log were 112.4 and 2436.6. The third log was a message log that indicates a communications failure occurred. The last entry is from status channel 1 and indicates that the switch closed (the value is 1).

13.9 LOG ERRORS

If an error occurs when writing the logs from memory to disk, the writing will be stopped to prevent further problems and the current log task will process a special file called LOGERROR.RTU. A common cause of error is a full disk device. The LOGERROR file can do whatever is necessary to change log files to another disk, alert an operator, or simply clear up some disk space by deleting files. The LOG EVERY xx command can be used within the file to restart the logging process after freeing up some disk space. To alert the operator, a channel can be setup (such as a spare status channel not associated with any real I/O) and forced into alarm from within the LOGERROR file.

13.10 USE OF LOG FILE DATA IN EXTERNAL PROGRAMS

When a log file is displayed from within the RTU program, it is "unwound" from the compressed format into a text line that is similar to that of the normal screen or report text. This is because the RTU program can look up the channel name and units from its internal setup tables and restructure the line accordingly. External programs, such as Lotus or Dbase, will have to provide their own equivalent function in order to regenerate the text line. The value data, however, is always stored in engineering units so it can be used without further conversion.

13.11 LOGGING TO A PRINTER

The Log system allows for any valid DOS file to be specified as the destination for all log saves. Therefore, the system printer can be accessed as a valid DOS device by specifying PRN as the name of the file with the command line LOG FILE PRN. The logger treats this as a special case and formats the output to the printer much as it appears on the screen. If the PRN device is used as the log file, then the logger will write directly to the printer rather than sending the log data to the disk file. The printout occurs at the same time as the save to disk would have occurred.

Note that each RTU has its own LOG file name. Therefore it is possible for one RTU to print directly to the printer, while another goes to a file for later display or printout.

Printing log data is done with the LOG TO PRN option to the logging command. This will pull data from the existing memory and disk file log and format it into text strings that are sent to the printer. For example, the line LOG FOR a1:A16 to PRN will cause all existing log entries for analog points 1 through 5 to be sent to the printer in a readable text form.

14 AUTOMATIC SCADA FUNCTIONS

The SCADA system is designed to operate in an unattended environment and must handle fairly complex tasks on a routine basis. The TEST SCADA program has the ability to automatically perform a number of different tasks in response to certain events. These tasks are designed to be useful in applications such as industrial monitoring as well as simple process control. For most events, the program's response is programmable on a point by point basis.

Also described here is the SCADAWARE Image Save function. This provides an automatic disk based data save facility that allows fast storage of the program's internal data structure, or its "Image." This disk file allows for rapid recovery from computer failures, and also allows network access to SCADAWARE information by other computers with access to common disk drives.

14.1 ALARM SEQUENCING

Any channel can be configured as an alarm channel. The complete alarm process for a channel is made up of several stages which are as follows:

No Alarm	- A point in its normal state or within alarm limits.
Timing	- Out of range but not long enough.
New Alarm	- Out of range and not acknowledged.
Alarm	- Out of range and acknowledged.
Deadband	- Back in normal range, but within Deadband (value channels).
Reset	- Back in normal range (and outside Deadband).

The alarm status of a channel is displayed with a flag in the alarm column on the right side of a channel display. The flags displayed for the different alarm stages are:

No Alarm	- (nothing)
Timing	- Tim
New Alarm	- New
Alarm	- Alm
Deadband	- DB
Reset	- Res

The No Alarm stage is the stage in which a status channel is in its normal state or a value channel (or any derivative channel type) is within its alarm limits (and the channel has been reset since any previous alarm). If these conditions change, the Timing stage of the alarm process is activated. The Timing stage is a period of time that the computer will wait before putting a channel into the New Alarm stage. If a channel returns to its normal state during this period of time, no alarm will be generated and the channel will be returned to the No Alarm stage. If, however, the timing period expires and the alarm condition still exists, the channel will go into the New Alarm stage.

The length of time the computer waits in the Timing stage is configurable for each channel with the Alarm Delay Secs option. If this option is set to 0 the Timing stage will be eliminated and a channel will go from No Alarm to New Alarm immediately upon detection of an alarm condition.

A channel will remain in the New Alarm stage until it is acknowledged. A channel can be acknowledged by pressing the F2 key while the channel is displayed or by using the ACK command. If the channel's alarm condition has cleared by the time the channel is acknowledged the channel will go into the Reset stage. If the alarm condition has not cleared, the channel will go into the Alarm stage after being acknowledged. When the alarm condition finally clears, the channel will then go into the Reset stage.

For value derivative type channels only, a Deadband option is available during channel configuration. The Deadband is used to alter the conditions that determine when an alarm is actually cleared. The Deadband

affects both the high and low alarms, but in opposite ways. The high alarm is initiated whenever the channel reaches the high value (or above) and the alarm delay times out. After entering the alarm state, the high alarm will not clear until the channel value drops to the high alarm minus the Deadband. The low alarm works in just the opposite way. The low alarm does not clear until the channel value rises above the sum of the low alarm and the Deadband.

[alarm sequence slide]

The Deadband parameter is provided to reduce nuisance alarms on channel types that hover near the alarm point. Without the Deadband, the channel would continually re-alarm in response to small changes in the channel value (if Reset After Ack is set ... explained below). The Deadband allows for customization of each point to determine how much of a change is required to clear an existing alarm. If a channel has a Deadband of 0 then this parameter is ignored.

If a Deadband is specified, a Deadband stage exists between the Alarm stage and the Reset stage. The Deadband stage indicates that the channel has cleared its alarm condition but has not cleared it by as much as the Deadband. A channel's value can fluctuate and cause the channel to switch between the Alarm stage and the Deadband stage. When a channel clears the alarm condition by the amount of the Deadband the channel will go into the Reset stage.

The Reset stage indicates that a channel has gone into alarm, been acknowledged, and returned to its normal condition. Before a channel can generate another alarm it must first be reset. A channel can be reset by pressing the F3 key or by using the RESET command. Resetting a channel will complete the alarm cycle and return a channel to its No Alarm stage. Each channel has a configuration option called Reset After Ack. If this option is set a channel can automatically reset itself after it is acknowledged if its alarm condition has cleared. If the alarm condition has not cleared at the time of the acknowledgement, the channel will remain in the Alarm stage. When the alarm condition is cleared the channel will then skip the Reset stage and return to the No Alarm stage.

14.2 ALARM AND ABNORMAL CONDITION RESPONSES

A primary function of the SCADA system is to respond to abnormal conditions. The exact response is programmable for each individual channel using the PROG or CONFIG command. The options available for each channel regarding the response to an abnormal condition are explained below.

ALARM TYPE CHANNEL? This determines whether or not a channel will go into alarm when an abnormal condition occurs. If this option is set, the channel will go through the alarm sequence explained above every time an abnormal condition occurs. As the alarm state of a channel changes, the display for that channel will change in color or blink status. Setting this option will also allow a First Out alarm to be logged (see section 13.6 for more information about First Out alarms) and possibly a local horn to be blown. This option does not have any effect on any of the other responses to abnormal conditions. Note that the external horn is determined by the SET HORN command where any channel can be designated as the physical horn output.

BLOW HORN ON ALARM? Setting this option will allow a local horn to be blown whenever an abnormal condition occurs for this channel. However, other conditions must also be met for the horn to work. First, the channel must also be setup as an alarm type channel using the setting described above. Second, a global variable called WARBLE must be set for the horn to work. The WARBLE setting is controlled by the SET WARBLE command. If WARBLE if OFF the horn can not sound. Use the command SET WARBLE ON to allow the horn to sound.

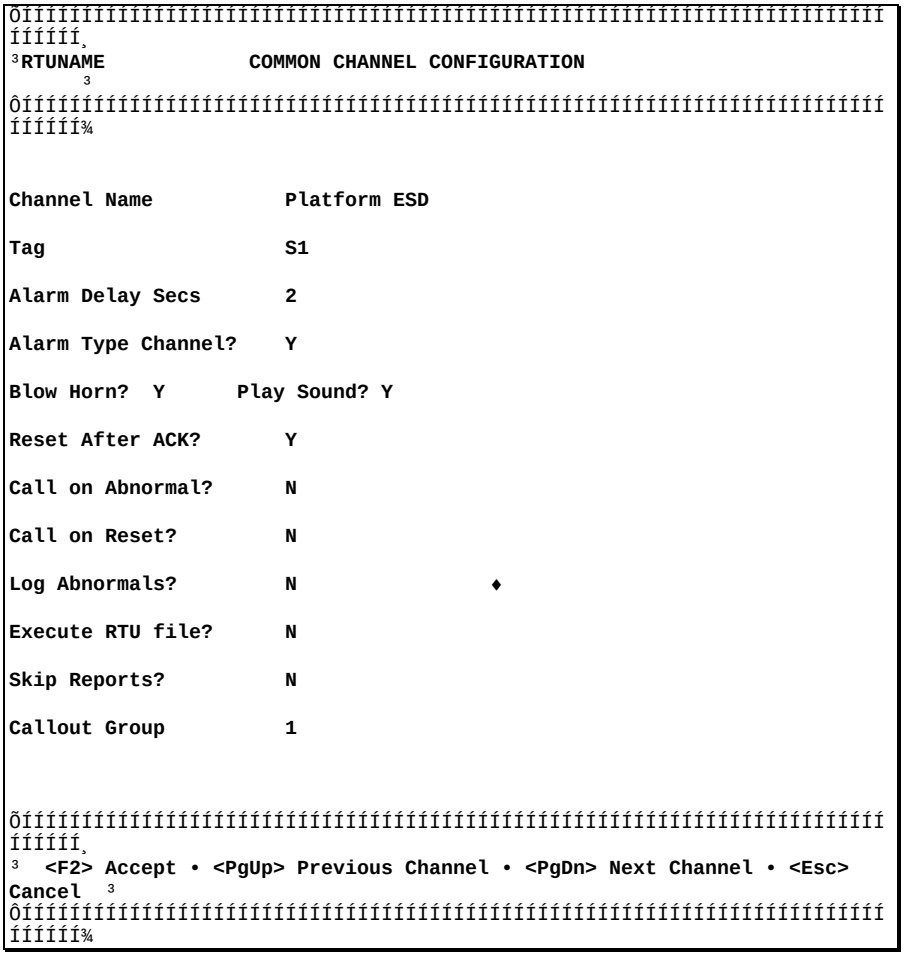
CALL ON ABNORMAL? If this option is set for a channel, the process to activate all links that belong to the call out group associated with that channel will begin when an abnormal condition occurs.

CALL ON RESET? Setting this option will cause different effects depending on whether the channel is setup as an alarm type channel or not. If not an alarm type channel, the process to activate all links that belong to the call out group associated with that channel will begin when the transition is made from an abnormal to a normal state. If setup as an alarm type channel, the links associated with the channel's group will not be activated until the channel returns to a normal value and the channel is acknowledged. This is the time that the channel enters its RESET state (meaning that it is waiting to be reset). If the channel is configured for AUTO RESET, the channel will be reset at the same time it is acknowledged. Again, this is the time that the call out process will begin.

LOG ABNORMALS? Setting this option will cause the time, date, and channel value to be sent to the program's logging system at the time the abnormal condition occurs.

EXECUTE RTU FILE? Setting this option will allow a command file (or library procedure) to be executed each time the channel enters an abnormal state. The name of the command file (or procedure) must be the same as the channel tag. Refer to section 8.4 of this manual for more information about command file execution on abnormal conditions and return to normal conditions.

PLAY SOUND? This option controls playback of a pre-recorded message using the PLAY function on the utility task. The utility task is sent the message PLAY RTUNAME.TAG. The PLAY command performs just as if the command had been entered by hand or in a command file.



14.3 COMMUNICATION RESPONSES

The nature of SCADA systems is that many events are based on the activities of a communications line. Each RTU task can be set up with individual responses for initial communications connection, disconnection, and failure to connect. The command sequence associated with each event is individually programmable via a standard command file.

The related files are called CONNECTx.RTU, BYEx.RTU, and COMFAILx.RTU, where the x is replaced with the task number executing the file. For example, task 2 executes CONNECT2.RTU whenever it detects a connection to another unit. Example uses of these files include such things as time-stamping all connections with the LOG command or sounding a horn with the HORN ON command in the event of a communications

failure.

When a task is processing a callout because of a communications link activation, it will look for and process a file associated with the link when a connection is made. This processing follows the normal processing that the task does on any connect. The name of this file is LINKx.RTU, where the x is replaced with the link number. For example, a task handling link number 3 will execute "LINK3.RTU" when a callout attempt is completed successfully. For more information on theses files and other files that automatically get processed, refer to the TSP COMMAND FILES, section 8 of this manual.

14.4 TIME INTERVAL PROCEDURES

Many systems require processing at regular time intervals. This type of processing can be performed by using Timer channels. For example, polling can occur on a regular basis by having a timer channel set up to call on abnormal. When the timer reaches its low alarm value (if a down counting timer) it will activate all links contained in the timer's callout group. Command files can also be processed periodically by using Timer channels setup to execute command files when abnormal conditions occur.

After the call out is made or the command file is processed the timer channel must be reset to begin timing again. One way to do this is to set the AUTO RESTART option when configuring the timer channel. If this is set, the timer will automatically be reset to its high alarm setpoint value whenever it reaches its low alarm value (assuming again that it is a downward counting timer). It will automatically begin timing down again from that value. The same procedure will work if an upward counting timer is used. The only difference is that the channel would be reset to its low alarm value when it reaches its high alarm value.

A second way to restart the timer is to use the CALC command in one of the files that get processed. For example, if a timer channel was used to activate a link, the CALC command could be used in the BYE file to restart the timer automatically when the call out is complete. For command files that get executed when a timer times out, the CALC command could be used at the of the file to restart the timer after the rest of the file is processed.

14.5 DAILY PROCEDURES (AGENDA)

Many systems require processing at specific times of day. The best way to handle these requirements is by placing the steps required in command files and having these files processed by the UTILITY task. The processing of these files would then be initiated by the AGENDA system.

The AGENDA system monitors the time of day and processes commands according to the clock. For example, the AGENDA can have a host call an RTU at a specific time of day, perform a download, and generate a report. The AGENDA system has the ability to send commands to any RTU task at any time of day, and reset itself at midnight so that the process repeats each day.

[agenda slide]

One typical AGENDA activity is to have the program read the computer's "watch" once per day (say at 2:00 p.m.) to synchronize the DOS software clock with the more accurate digital timekeeper. It is not unusual to see lapses of a minute or so each day, and reading the watch will restore the system's time of day to the more accurate time base. This is only necessary on PC based systems that do not have a standard CMOS type battery powered clock device. IBM-AT (286) and better systems have a standard CMOS clock that is automatically read throughout the day to avoid variations in the real time clock. Therefore, daily setting of the clock should only be needed on PC (8086) level systems.

Another typical procedure is to transfer accumulated totals to value channels from totalizer or counter channels. After the transfer, the source channels are set to 0 to begin another day's accumulation. Both totalizers and counters have a time and date associated with them that indicates when the channels were last cleared. This current time and date is set into these channels whenever they are set to exactly 0. A typical daily file is as follows:

```
; Daily file for GA343A
calc ygastot = q1  ; pick up sales total and put in yesterday gas
calc q1 = 0   ; clear the gas totalizer
calc yoiltot = c1+c2 ; sum up both oil meters and save
calc c1:c2 = 0   ; clear the oil meters
image save    ; save the value channels
```

In this case, the channels YGASTOT and YOILTOT are assumed to be value channels set up to store the yesterday production values. The last line of this file will cause all channel values (along with other data) to be saved to disk as soon as possible. This data can be read when the system starts up (perhaps by a line in the START0 file) so that the most recently saved data can be restored to the channels.

When creating an agenda list, make sure the commands for each task are listed in order of time. The computer will not automatically sort the list for you. See the AGENDA command in the TEST SCADA PROTOCOL COMMAND REFERENCE MANUAL for more information about the Agenda system.

14.6 AUTOMATIC DATA IMAGE SAVE

This section provides information on how TEST's SCADAWARE saves data in a real time disk based image file. The file has several purposes. It allows instant loading of all important data when the program is initially started from the DOS command prompt. It also allows network operation by putting all system data into an accessible disk file which can be used by other copies of the program running on other computers on the network.

The Image Command options will be of use to all program users. Although the inner details of the Image File are provided here, it is not anticipated that they will be required by the user. TEST is making this information available for completeness and to allow advanced users access to the inner workings of the image system.

The program has an automatic, transparent data save system that continuously saves many dynamic values while the programming is running. This feature saves an image of certain data to a file using the actual internal format of the program. Normally, the program saves values in text format so they can be interchanged with other programs or edited with a text editor. The image, however, uses the internal format because it requires no data conversion. This allows for the fast performance of the image system, making its use transparent in most applications. Because of its format, the image file can only be used by the TEST SCADA program for the sole purpose for which it is intended.

This automatic save feature provides a continuous backup of the program's data on disk. Once a file has been created by the program it can be read back in at a later time, either by the same system or another system running the same software. On most systems, this file is loaded automatically upon startup if it exists. This provides a quick and easy way for a program to resume operation with the values it had when it was last running. On network systems where several units must contain copies of the same information, each unit can continuously be saving its own local data to a file while other units are continuously retrieving a copy of that data

from the same file.

14.7 SCADAWARE LITE IMAGE INSTRUCTIONS

Note: The extensive Image features listed here apply only to the full feature SCADAWARE program, RTUMON3, and not to SCADAWARE LITE, program RTULITE3. RTULITE3 has a simple image system which only allows saving and loading of all data at one time. RTULITE3 has only one image file, while the full version has four simultaneous files. Therefore, RTULITE3 can only use the following basic functions:

Image Load
Image Save
Image on xx
Image file fname
if @image(0)

These functions are described in detail below.

14.8 IMAGE FILE STRUCTURE

The Image file is a random access DOS file with a size that is fixed to accommodate the particular requirements for each system configuration. The Image represents a direct binary copy of what the computer holds in RAM memory. The binary image allows very fast save and retrieve processes because no data conversion is required to access the data. The program simply copies the data to and from the disk at the highest possible speed.

Within the program similar types of data are grouped together in tables. When a command is processed to save an image of the program's data, the information is taken directly from these tables and written to a file. Although the size of each file will vary depending on the setup of each system, the internal layout of all image files is the same. Following is a list showing the internal layout of all data stored in an image file:

1 - IMAGE HEADER
This contains the image version number, link count, RTU count, and channel count for each type of channel. This also contains the file location for each group of data stored in the file.

2 - LINK OBJECTS
3 - FLAG VARIABLES (Both Global and RTU Flags)
4 - UPDATE TIME AND DATE FOR EACH RTU (Except RTU 0)
5 - DEFAULT LINK FOR EACH RTU
6 - SYSTEM VARIABLES
This contains the printer port number, the number of ticks to wait between printing each page, and the number of lines per page for a report

7 - STATUS CHANNEL DATA
Alarm Delay Counters
Alarm States (in alarm, need reset, ...)
Alarm Modes (call on abnormal, auto reset, ...)
Channel Status (enabled, on hold, remote)
Current Values
Channel Objects

8 - OUTPUT CHANNEL DATA
(Same as shown for Status channel data)

9 - ANALOG CHANNEL DATA
Alarm Delay Counters
Alarm States (in alarm, need reset, ...)

Alarm Modes (call on abnormal, auto reset, ...)
Channel Status (enabled, on hold, remote)
Current Values
Low Alarm Setpoints
High Alarm Setpoints
Channel Objects

10 - PID CHANNEL DATA
(Same as shown for Analog channel data)
|
|
16 - VALUE CHANNEL DATA
(Same as shown for Analog channel data)

When a save is done for the first time which causes the image file to be created, the entire list of data that can be saved to the image file is actually written to the file. Once all data has been saved, the image file is closed. Closing the file will allow the correct file location and size to be set in the File Allocation Table (FAT). During the time of the first image save no task switching is allowed. This is to prevent any problems from occurring should another task try to access or create the same file at the same time.

Once a file is properly defined in the FAT, multiple tasks can access it at the same time. This means that task switching does not have to be (and is not) suspended during subsequent image saves. Likewise, task switching is not suspended when data is being loaded from an image file. Therefore, more than one task can access a single file at the same time without interfering with one another. This can be extremely useful on network systems where one task is regularly saving an image of the program's data to a file and another task is reading the saved data from the file and sending it to another unit on the network.

As mentioned, all data must be written to the image file when the file is first created. However, subsequent saves have the option of saving all data or just select groups of data. This allows dynamic data to be saved very frequently without having to save static data as well each time. For example, the image system can be setup to save frequently changing data, such as channel values, every couple of minutes or so. The system can also be setup to save rarely changing data, such as link data, once an hour or so. The ability to select certain groups of data to be saved also applies to data loads. This will keep the system from being bogged down needlessly saving or loading data that rarely changes.

14.9 BASICS OF IMAGE OPERATION

Up to 4 files can be open at the same time for saving or loading an image of the program's data. The status of each file is totally independent of the others. Within the program exists a table which keeps track of these four files and is responsible for managing the "image" system. This table is known as the "image table" and it looks something like this:

IMAGE TABLE
[1] File Name
File Status (Open or Closed)
Task Number
Automatic Mode (Yes or No)
Seconds to Delay
Seconds Left
Error Code
Message
[2] ...

[3] ...

[4] ...

When the program is first started this table is filled with default values. The file name for each position in this table is set to the system name with the extension "IMG". The system name is determined by the NAME command in the main configuration (DAT) file. Therefore, a system named HOST would contain the file name HOST.IMG for each position in the image table.

Other default values for each position in the image table are:

File Status	-	Closed
Task Number -	-	-1
Mode	-	Manual
Seconds to Delay	-	60
Seconds Left	-	0
Error Code	-	0

The only item in the image table that has a different default value for each position in the table is the MESSAGE. The default messages for each position in the image table are as follows:

IMAGE TABLE
- [1] Message - IMAGE SAVE VALUE FOR 1
- [2] Message - IMAGE SAVE VALUE FOR 2
- [3] Message - IMAGE SAVE VALUE FOR 3
- [4] Message - IMAGE SAVE VALUE FOR 4

Each RTU type task has associated with it an index number which represents a position in the image table. By default, the index number assigned to each task during startup is 1. Any time a task processes an IMAGE command it will apply to the data in the image table at the position for which its index number corresponds. For example, if a task's current index number is 2 and the task processes a command to change the image file name, only the image file name for image table position 2 will be changed.

With all of this in mind, let's look at the basics of how the image system is designed to work. The image system is designed to save/load data to/from disk either on command or automatically. In automatic mode, the image system will save data to disk at regular intervals. In this mode, the image system is responsible for counting down a timer, sending a command to the right task to save the data when the timer has reached 0, and resetting the timer to start the cycle over again. When not in automatic mode, the command to save an image of the program's data must be typed in by the user or processed in some other way that is external to the image system itself.

Typically, if the image system is going to be used at all, it will be setup to operate in automatic mode. To illustrate how this is done, let's examine the simplest case where one task will be used to save all data to the image file on a regular basis, say every 5 minutes. For purposes of this example, let's assume that the system name is HOST and that we will use the default image file name to save the data. Now, the first thing we must do is decide which task will be responsible for actually saving the data to the disk. A good candidate for this would be to use an RTU type task that is not dedicated to communications with other units. For our example, we will choose to setup task 3 as a Utility task which will be used to actually save the data.

To begin the automatic save process we only need to process the following two commands:

```
IMAGE TASK 3
IMAGE ON 300
```

Because the image table index number of each RTU type task is set to 1 upon startup, these two commands can be processed by any RTU type task. No matter which task processes the commands, they will be processed with respect to the first position in the image table. Before these commands are processed the first position in the image table would look like this:

```
IMAGE TABLE
    [1]     File Name               HOST.IMG
```

	File Status	Closed
	Task Number	-1
	Automatic Mode	No
	Seconds to Delay	60
	Seconds Left	0
	Error Code	0
	Message	IMAGE SAVE VALUE FOR 1

After the commands are processed the first position in the image table would look like this:

IMAGE TABLE

[1]	File Name	HOST.IMG
	File Status	Closed
	Task Number	3
	Automatic Mode	Yes
	Seconds to Delay	300
	Seconds Left	300
	Error Code	0
	Message	IMAGE SAVE VALUE FOR 1

The Seconds Left counter would then begin counting down. When it reaches 0 the Message, IMAGE SAVE VALUE FOR 1, will be sent to task 3 and the Seconds Left counter will be reset to 300 to restart the cycle. When task 3 processes the IMAGE SAVE VALUE FOR 1 command, it will save all channel values to the file specified by position 1 in the image table, namely HOST.IMG.

Now, let's consider a more complicated example where image table position 2 will be used to save Value, Counter, and Timer channel values to the image file on a regular basis, say every 3 minutes. Again, we will assume the system name is HOST and that we will use the default image file name. We will also assume task 3 is a Utility task which will be used to actually save the data.

To begin the automatic save process we would have to process the following commands:

```
IMAGE SEL 2
IMAGE TASK 3
IMAGE MSG IMAGE SAVE FOR 2 VALUE VCT
IMAGE ON 180
```

Although these commands can be processed by any RTU type task, they are generally put in a startup file which automatically gets processed by task 0 when the program is started. Before these commands are processed the second position in the image table would look like this:

IMAGE TABLE

[2]	File Name	HOST.IMG
	File Status	Closed
	Task Number	-1
	Automatic Mode	No
	Seconds to Delay	60
	Seconds Left	0
	Error Code	0
	Message	IMAGE SAVE VALUE FOR 2

After the commands are processed the second position in the image table would look like this:

IMAGE TABLE

[2]	File Name	HOST.IMG	File Status	Closed
	Task Number	3		
	Automatic Mode	Yes		

Seconds to Delay	180
Seconds Left	180
Error Code	0
Message	IMAGE SAVE FOR 2 VALUE VCT

Again, the Seconds Left counter would then begin counting down. When it reaches 0 the Message, IMAGE SAVE FOR 2 VALUE VCT, will be sent to task 3 and the Seconds Left counter will be reset to 180 to restart the cycle. When task 3 processes the IMAGE SAVE FOR 2 VALUE VCT command, it will save the current values for all Value, Counter, and Timer channels to the file HOST.IMG.

Note that the command IMAGE SAVE FOR 2 VALUE VCT, which is used to save the Value, Counter, and Timer channel values, contains the parameters FOR 2 within the command. These parameters are used to instruct task 3 to process the command IMAGE SAVE VALUE VCT with respect to position 2 in the image table. If the parameters FOR 2 were not specified, task 3 would process the command with respect to its currently selected image table position. This will not cause any problems if position 2 in the image table is currently selected for task 3. However, if position 2 is not currently selected for task 3 and the FOR 2 parameters are not specified in the command, the image save would be attempted for another position in the image table and its associated file. Chances are this will not have the desired results. Therefore, it is always best to specify the desired image table position within the command using the FOR option.

14.10 IMAGE COMMAND

The saving and loading of data in an image file as well as the image system setup is controlled through the use of the IMAGE command. The format of this command is the word IMAGE followed by a keyword and a list of parameters. This command can be processed by any RTU type task. As mentioned before, there can be up to 4 files open and operating independently at the same time for saving or loading an image of the program's data. The management of these files is done through the use of an internal image table. Each RTU type task has a number associated with it which represents an index to the image table. By default, the processing of any IMAGE command is applied to the position in the image table for which the task's index number is current.

As an alternative, the special keyword FOR followed by a number can be used in the IMAGE command to specify a position in the image table for which the command should be applied. For example, suppose that the current image table index for task 0 is 1. If task 0 were to process the command

IMAGE FILE HOST

the file name for position 1 in the image table would be set to HOST.IMG. On the other hand, if task 0 were to process the command

IMAGE FILE FOR 2 HOST

the file name for position 2 in the image table would be set to HOST.IMG. By specifying FOR 2 in the command, the task's current image table index of 1 is overridden by the number specified in the command, namely 2. This would cause the command to affect position 2 in the image table rather than position 1.

If the FOR keyword is to be used in the IMAGE command it must always appear as the 3rd item on the line and be followed by a number in the range 1-4. If not, it will be ignored and assumed to be a parameter used by one of the other keywords.

A list of keywords and parameters that can be used by the IMAGE command are given below. All parameters are shown in brackets to indicate that they are optional. For some keywords, default values will be assumed for the parameters if they are not specified. For other keywords, the current value will be displayed if the task processing the command is task 0. Otherwise, the command is ignored. If no keyword is specified in the IMAGE command the keyword DUMP is assumed.

ON [xx] This will turn on the automatic feature to continuously save or load an image of the program's

data. When this is ON, a message is sent to a task at regular intervals. The message that is sent and the task that it is sent to are obtained from the position in the image table for which the IMAGE ON command was processed. The optional parameter allows the interval to be set at which the message is to be sent. This number should be specified in seconds. If this parameter is not specified, the program will assume the value that was used the last time this feature was activated. The default value upon starting the program is 60 seconds.

Examples:

```
IMAGE ON              ;start auto mode using default time
IMAGE ON 30           ;start sending msg every 30 seconds
IMAGE ON FOR 2        ;start for image table position 2
IMAGE ON FOR 3 45     ;start for position 3 every 45 secs
```

OFF — This will turn off the automatic feature to send an image message to a task at regular intervals. This will only affect the position in the image table for which the command was processed.

Examples:

```
IMAGE OFF             ;quit auto mode for current position
IMAGE OFF FOR 3       ;quit auto mode for image table position 3
```

SAVE [data] [channel types]

This is used to save data to the image file. The parameters are used to specify exactly what data will be saved. All data is saved to the image file the first time a save is done and an image file is first created, regardless of which parameters are specified. If no parameters are specified, the keyword ALL is assumed and all data will be saved. The optional keywords which can be used to specify which data to save are listed below.

ALL	All data that can be saved in image the file
SYSTEM	All System Data. This includes links, flags, RTU update times and dates, default RTU links, and system variables.
LINKS	Link Objects
FLAGS	Flag Variables (Both Global and RTU Flags)
UPDATE	RTU update times and dates
RTULINKS	Default RTU links
VARS	System Variables
ADELAY	Channel Alarm Delay Counters
ASTATE	Channel Alarm State (in alarm, need reset, ...)
AMODE	Channel Alarm Mode (call on abnormal, auto reset, ...)
CSTATUS	Channel Status (enabled, on hold, remote)
VALUE	Current Channel Values
LOALM	Channel Low Alarm Setpoints
HIALM	Channel High Alarm Setpoints
CONFIG	Channel Objects

Following the keywords which specify which channel data to save is another optional parameter which can be used to specify exactly which channels to save the data for. This parameter can be a single letter or a list of letters grouped together, where each letter represents a particular type of channel. The letter to specify for each type of channel is as follows:

S - Status Inputs
O - Status Outputs
A - Analog Inputs
P - PID Channels
M - AGA3 Channels
Q - Totalizer Channels
C - Counter Channels

F - Function Channels
T - Timer Channels
V - Value Channels

If this parameter is not specified when saving channel data, the particular data specified will be saved for all channel types. Whenever a save is done for any type of channel, all channels of that type on the system are saved. You can specify data for a particular type of channel to be saved, but not a range of channels within that channel type.

NOTE: When data is saved to an image file there are several checks that occur before the save is actually performed. First of all, a check is done to see if the image file is currently open (it could have been left open from a previous IMAGE SAVE or IMAGE LOAD command). If the file is not currently open, it is opened at this time. Second, a check is done to see if a save has been done since the file was opened. If so, the save is performed at once. If not, a check is done to see if the image version number, number of links, number of RTUs, and number of channels defined in the program's setup match those contained in the header of the image file. If all of these numbers match, the save is done at once. However, if there is any difference in these numbers, the current image file is automatically deleted and recreated using the current data. All data that can be saved in the image file is written to the file at that time, regardless of which parameters were specified in the IMAGE SAVE command.

Examples:

```
IMAGE SAVE
IMAGE SAVE ALL
IMAGE SAVE SYSTEM
IMAGE SAVE LINKS
IMAGE SAVE CSTATUS          <--     Status of all channels
IMAGE SAVE CSTATUS SO       <--     Status of Status Inputs and Outputs only
IMAGE SAVE VALUE            <--     Values for all channels
IMAGE SAVE VALUE C<--       Counter channel values only
IMAGE SAVE VALUE MQCT       <--     Values for Meters, Totalizers, Counters, Timers
IMAGE SAVE CONFIG           <--     All channel objects
```

LOAD [data] [channel types]

This is used to load data from the image file. The parameters used here are the same as the SAVE option described above. Just as with the SAVE option, either all data or just specific groups of data can be loaded from the image file at one time. Refer to the SAVE option for details about available parameters.

A check is done prior to the actual loading of data from an image file. The purpose of the check is to make sure the data contained in the image file can be directly loaded into the program's variables. Data can not be loaded if there is not an exact correlation between the data contained in the file and the program's variables.

The check begins by reading the image header from the image file. This image header contains an image version number which represents the exact layout and position of all data stored in the file. If this version number does not match the program's internal image version number, the entire load is canceled.

The remaining checks compare the number of links, RTUs, and channels stored in the image file against the number actually defined in the program. If any of these numbers do not match, the corresponding data in the file can not be loaded. However, the entire load is not canceled.

For example, if the program contains 10 links but data for only 8 links are stored in the image file, the link objects can not be loaded from the image file. Likewise, if the program contains 5 RTUs but the image file contains data for 6 RTUs, no RTU type data can be loaded from the image file. This includes flags, update times and dates, and default RTU links.

Last of all is the check for the correct number of channels. Consider the situation where a system that contained a set number of channels performed an image save and created an image file. Now assume that the

program was stopped, a few more channels were added (say 5 more Counter channels), and the program was restarted. If the program then tried to do an image load of all data from the existing image file, the Counter channels could not be loaded. All data for all other channel types could be loaded, but no data for any counter channels could be loaded. This is because the number of Counter channels contained in the image file does not match the actual number of Counter channels contained in the program.

Remember, only incompatible image version numbers can completely prevent an image file from being loaded. Other incompatibilities will only prevent parts of the image file from being loaded. In situations where the data stored in the image file does not properly correspond with the program setup, an IMAGE SAVE can be performed to correct the problem. Whenever an image save is done for the first time since the image file was opened, the system setup is checked with the header in the image file. If there are any differences, the image file is automatically deleted and a complete image save is done, regardless of the parameters specified in the IMAGE SAVE command.

Usually, when the program is first started an IMAGE LOAD command is processed to load in all data that was current at the time the program was halted. Then, an IMAGE SAVE is processed automatically every so many seconds to generate a continuous backup of the current data. If a change has been made in the system setup between the time the program was stopped and restarted, the data pertaining to the changes will not be able to get reloaded when the program is started and the IMAGE LOAD command is processed. However, the rest of the data can be reloaded. Once an IMAGE SAVE is done, the program will automatically fix the image file so that the header in the image file and the program setup are back in sync.

Examples:

```
IMAGE LOAD
IMAGE LOAD ALL
IMAGE LOAD SYSTEM
IMAGE LOAD LINKS
IMAGE LOAD CSTATUS          <--     Status of all channels
IMAGE LOAD CSTATUS SO       <--     Status Inputs and Outputs only
IMAGE LOAD VALUE            <--     Values for all channels
IMAGE LOAD VALUE C <--      Counter channel values only
IMAGE LOAD VALUE MQCT
IMAGE LOAD CONFIG           <--     All channel objects
```

FILE [file name]

TASK [task number]

This will set the task number for a single position in the image table. The task number that is specified must be an RTU type task. By default, all task numbers in the image table are set to -1. If a position in the image table attempts to process a message when the task number is -1, the program will send the message to the Utility task if it exists. (The Utility task can be defined by setting a task's ID to UTIL using the command

TASK xx ID UTIL

where xx is the task number.) There are 4 ways that a task number can be specified in the IMAGE TASK command. They are:

1. Actual task number
2. RTU name - look up the current link for the RTU and then the current task for that link
3. Task ID
4. Task keyword - LOCAL, UTIL, etc.

Examples:

```
IMAGE TASK 3
IMAGE TASK SS180            ;RTU name
```

IMAGE TASK UTIL
IMAGE TASK FOR 2 3 ;set task number for position 2 to 3

SEL [index number]

This keyword is used to set a task's index number which corresponds to a position in the image table. The specified number must be in the range 1-4 in order for the command to be successful. Before assigning a new index number to a task, the position in the image table associated with the task's current index number is closed. Closing the position typically involves turning off the automatic save feature and closing an open image file. However, if the same file is open and being used by another position in the image table, the file will not physically be closed but rather a flag will simply be set to indicate that the file is no longer in use by this position. For more details, refer to the section below called CLOSING AN IMAGE FILE.

Examples:
IMAGE SEL 2
IMAGE SEL 3

MSG [xx]

This will set the message that is sent to a task when using the automatic image feature. If this command is processed by task 0 and no parameter is specified, the current message for task 0's selected image table position is displayed.

In situations where only select data is to be saved or loaded it might require several image commands. For example, assume that you want to continuously save only the current channel values, alarm states, and alarm delay counters for all channel types. To do this would require the following 3 image commands:

IMAGE SAVE VALUE
IMAGE SAVE ASTATE
IMAGE SAVE ADELAY

Since each image table position has only 1 message associated with it, the image system can not be setup to directly process these 3 messages automatically. However, these 3 messages can be put into a file which the image system can process automatically using a single command. In essence, this gives the image system a way to indirectly process multiple messages automatically. For example, assume that the 3 messages shown above were put into a file called IMGSAVE.RTU. To process these 3 messages automatically the image message would simply have to be set to READ IMGSAVE.

Examples:

```
IMAGE MSG IMAGE SAVE              ;set msg to be IMAGE SAVE
IMAGE MSG FOR 2 IMAGE SAVE        ;set msg for position 2
IMAGE MSG IMAGE SAVE FOR 2        ;position 2 specified in msg
IMAGE MSG IMAGE LOAD              ;set msg to load all data
IMAGE MSG IMAGE LOAD SYSTEM       ;set msg to load system info
IMAGE MSG READ IMGSAVE            ;set msg to read a file
IMAGE MSG                         ;display current message
```

DUMP

This keyword, which must be processed by task 0, will display the contents of the image table along with the image version number and the total number of links, RTUs, and channels currently defined.

Examples:
IMAGE
IMAGE DUMP

14.11 CLOSING AN IMAGE FILE

An image file is opened whenever data is first saved to it or loaded from it. If a save is initiated by the

automatic save feature the file will remain open after the save is complete. This is done to keep the file from having to be reopened each time the next save is done. However, if the save is initiated by a user command, the file will be closed after the save is complete. In the case of an image load, the image file is left open after the load is complete.

Once an image file is open, there are 4 ways it can be closed. They are:

1. Error - An image file will automatically be closed whenever an error occurs during an attempt to save data to or load data from the file.

2. IMAGE OFF command - This command will turn the automatic save feature off and close the image file.

3. IMAGE FILE command - This command will close the image file for a particular position in the image table if it is open and assign a new file name to that position.

4. IMAGE SEL command - This command will close the image file for the task's current position in the image table and set a new image table index number for the task.

Each position in the image table contains a file name and a flag which indicates whether that file is open or closed. When an attempt is made to close a file for a particular position in the image table, a check is done to see if another position in the table has the same file name and if the file is currently open for that position. If not, the file is physically closed and the flag is set for the current position in the table to indicate that the file is no longer in use by that position. However, if the file is currently open for another position, the flag for the current position is set to indicate that the file is no longer in use but the file will remain open. An image file can be physically closed only when the image table position attempting to close the file is the only position currently using the file.

14.12 LOCAL AND NETWORK IMAGE DATA

The image can be used several ways during both startup and normal program operation. The most basic application is to load the entire SCADAWARE setup from an existing image file. This lets the startup operating run much faster because all the text files used to describe a system do not have to be read. Instead, all the compiled binary data is accessed quickly from the image file. The following startup file segment illustrates the fast load feature:

```
image load
if @image(0) =0   ; test for validity of image load
 msg Image loaded properly
else
  msg Loading Individual Setup Files
  gosub $S.lin  ; load in the link information
  gosub rtu1.rtu  ; load in setups for rtu1
  gosub rtu2.rtu  ; load in setups for rtu2
  Image Save            ; Create initial Image file
endif
image on 30   ; start saving every 30 seconds
```

Note that the "IMAGE ON" line has a number specifying the number of seconds between saves. It is not practical to save the data constantly because that is all the system would have time to do. A compromise must be made between disk speed, processor speed, and how often data actually changes. For RTU systems with ram based disk drives, every 5 or 10 seconds is fine. For hard disk systems, every 30 seconds is a good starting point. Floppy based systems, which are very slow may have to use 5 minute intervals or longer.

If the image is periodically being saved as a result of the IMAGE ON command, a request will be made to immediately save the image whenever a SAVE, LINK SAVE or HALT command is issued. An image save can also be forced to occur at any time by processing the "IMAGE SAVE" command. This can be used after any

critical values change to ensure that the image contains the latest data. In either case, a request to save the image sets an internal flag that tells the One Second task to do the save as soon as possible.

Image data can be accessed into any SCADAWARE system with access to another system's Image file. Accessing a remote network system's image is identical to accessing a local drive. The only difference is the file specification of the data source. Instead of specifying a local drive, the remote system will specify the network drive which contains the image file. Subsequent image loads or saves will come from the network drive, effectively exchanging the specified data across units.

Network systems can load and save as much or as little as desired. It may be desirable to get channel data from the network drive, but save and store alarm status information relative to each system. In this case, each user will have its own set of alarm status conditions and setpoints. Alternately, all data can be retrieved from the network drive. Alarm status will affect that of the master station, rather then the local station. The exact setup will vary for each system depending on the needs and resources particular to each installation.

14.13 IMAGE DATA FILE STRUCTURE

Following is a description of all values saved in the data image.

All channels:
Tag , System point, RTU point, Owner, Channel type, New data, Alarm mode,
Alarm delay, Status mode - enabled, on hold, remote, Alarm state, Seconds till alarm

Status type channels: Normal state, Current status

Value type channels:
Low alarm value, High alarm value, Deadband, Input control channel,
Decimal places, Current value, Units

Status inputs: Latched status

Status outputs: Physical output, Physical off state

Totalizer channels:
Input channel, Minimum value, Factor, Start time, Start date, Started

Counter channels:
Threshold count, Counter factor, Filter ticks, Filter count, Raw counter
Start time, Start date, New raw data

Analog channels:
Offset condition, Value at 0%, Value at 100%, Negative condition
Raw value, New raw data, Slope, Offset

Function channels:
Input channel, Frequency factor, Sample seconds,
Channel type - difference, average, rate, function
Frequency accumulator, Frequency seconds left

Timer channels:
Output channel, Start channel, Auto output, Auto reset, Auto restart
Count direction - up or down, Timing state - on or off
Timer units - secs, mins, hours, days , Alarm value
Timer count, Active

Aga3 channels: Last error, Current error

Values saved for each RTU:
Time of last update
Date of last update

Values saved for each Link:
Link number
Number of task using the link
Maximum number of trys to call out when an attempt fails
Seconds to wait for a connect before giving up
Seconds to wait between call out attempts
Seconds left until call out is attempted
Flag indicating if unit is a host or not
Link status (idle, active, failed)
Number of callout attempts so far
Alternate link number

Global Variables that are saved:
Use Warble Use local speaker as an alarm horn?
Auto Backup Create backup files when using the test editor?
Has Printer Is a printer actually attached?
Page Eject Process form feeds at end of reports?
Printer Ticks # of ticks to wait between printing each line
Lines Per Page # of lines per page for reports
Use Alarm Delays Are alarm time delays in effect?
Do Reset Files Execute file when a channel returns to normal
Flag Arrays All predefined global and RTU dependent flags
Ticking Is ticker on?
Warbling Is warble on?

15 FUNCTION KEYS AND MENU SYSTEM

15.1 FUNCTION KEYS

The local CRT at the computer running the program has many abilities that the other remote terminals do not have. This is because the processor has direct access to both the local keyboard and the local CRT display memory. Remote terminals connect via a relatively slow speed serial interface that does not allow for function keys and pop-up menus.

The function keys (often called "F" keys) on the local keyboard are preprogrammed by the RTU/SCADA software to provide a convenient way to do often used program functions. Each key has a unique function during most program functions. However, when using the built in text editor the function keys take on a different meaning. When in the editor, holding down the ALT key while pressing a function key will do the same function that pressing a function key will do when not in the editor. For example, when not in the editor the F1 key can be used to silence an alarm. When in the editor, the ALT-F1 key combination can be used to silence an alarm.

The function keys of the local processor keyboard are monitored any time the program is waiting for keyboard input from the local user. This is any time the processor is not doing any other function. Although keys are stored as soon as they are hit, they are not read by the program until the command being executed is complete. This may cause a slight delay before the key is processed.

Hitting a function key causes one of two types of processing to occur. Some keys, like ACK and SILENCE, take immediate action by themselves. The actual program never sees these keystrokes. Other keys, like the MENU key, cause the keyboard processor to send a series of characters just as if the user had typed them in very quickly. This requires that the program be in a position to process these characters when the key is hit. If it is not, the program cannot properly process the function key action at that time.

You may notice some fast typing going on when you press a function key. It is common to see a "**CANCEL**" message show up just before the automatic command is typed. This is done by the function key processor in case the local task is in the middle of a display or the user is in the middle of typing in another command. If so, the display is canceled or the command being entered is aborted and the automatic command generated by the function key is processed.

The standard function key assignments are:

```
F1    Horn Silence
F2    Alarm Acknowledge
F3    Alarm Reset
F4    Rtu Channel Report
F5    Poll Current Rtu
F6    (Programmable)
F7    Display                 Sft-F7      Display First Alarm
F8    (Programmable)
F9    Start Main or User Menu       Sft-F9      Start User or Main Menu
F10   Select Rtu
```

Note that the F2 and F3 keys only affect points currently visible on the screen during a display. These keys do not affect points that are not visible. Therefore, it is necessary to go to each screen with a new alarm (or a resetting alarm) and ACK or RESET the alarms there. This eliminates missed alarms that were off screen when the function key was pressed.

The F7 Display key and the F9 Menu key are preprogrammed to perform special functions when used in conjunction with the **SHIFT** key. SHIFT-F7 will cause the system to locate new alarms that have not been acknowledged by the operator. The search will begin with the channel that first went into alarm for the current RTU. If no First Out alarm exists, the search will continue in the current RTU for new alarms using an internal channel order. If no new alarms for the current RTU are found, the system will scan other RTUs for pending

alarms. This allows an easy way to respond to a new alarm indication by having the program locate the new alarm. As each screen is acknowledged, SHIFT-F7 can be used to move to the next screen with a fresh alarm.

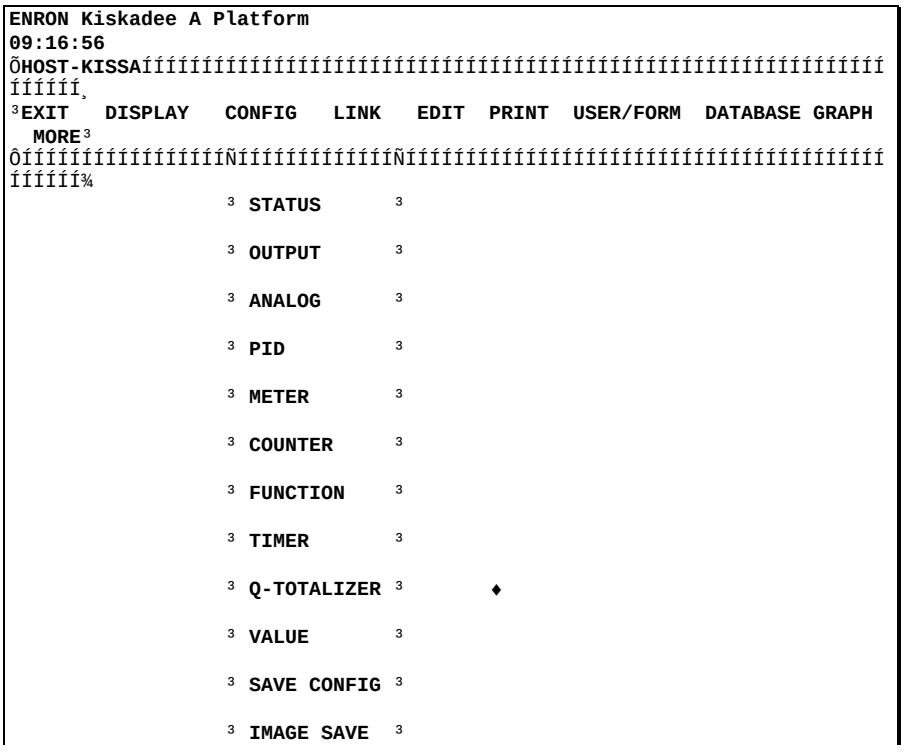
As mentioned earlier, the F9 key is used to put up a menu. There are two menu systems available and they are referred to as the MAIN menu and the USER menu. Normally, the Main menu is displayed when the F9 key is pressed. However, the SET MENU command can be used to change the default and have the User menu displayed when the F9 key is pressed. In either case, one menu is considered the primary menu and the other menu is considered the secondary menu. Just as the F9 key can be used to put up the primary menu, SHIFT-F9 can be used to put up the secondary menu.

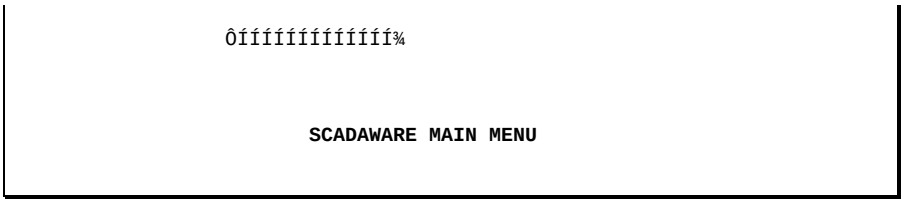
Except for the keys F7 and F9, all function keys can have a shifted-key action assigned to them. This is done by having Task 0 run a command file, with a separate file for each key. These files are named for "Shift F1, Shift F2" etc, and of course have the ".RTU" file type. The file names are SF1, SF2, SF3, ..., SF10 for each of the function keys. When the shifted function key is pressed, the system will look for the proper file, and if it exists, will send a message to Task 0 in the form of "READ SF1". The contents of the file are completely up to the user. Remember, the files SF7 and SF9 will not work because SHIFT-F7 and SHIFT-F9 have preprogrammed functions.

In addition to sending the command, the system also sends an escape and a Ctrl-C to Task 0. This will force the task out of any screen or menu so that the command can be processed.

15.2 MAIN MENU SYSTEM DESIGN

The SCADA program supports a main menu system for the local user that resembles a typical PC type "pull down" menu. This menu does not add any additional capability to the program itself. It simply makes it easier to do many of the common functions. The main menu is started by pressing the F9 key (or possibly SHIFT-F9), by entering the MENU command, or by pressing the right button of the mouse any time the local task is expecting keyboard input. Once up, the user can make selections from the menu by using the arrow keys and the [ENTER] key, or the mouse.





The main menu functions are divided into categories such as DISPLAY, CONFIG, and EDIT. Under each of these headings is a sub-menu that provides specific functions related to the main heading. For example, under DISPLAY is a list of channel types. Selecting the proper one will cause the program to put up a display just as if the F7 key or DISPLAY command was used.

One heading listed on the main menu is called USER. This item is used to display a custom menu which is uniquely designed for each location. The heading USER is used to differentiate the user's menu from the system's main menu. More on the user menu is explained shortly.

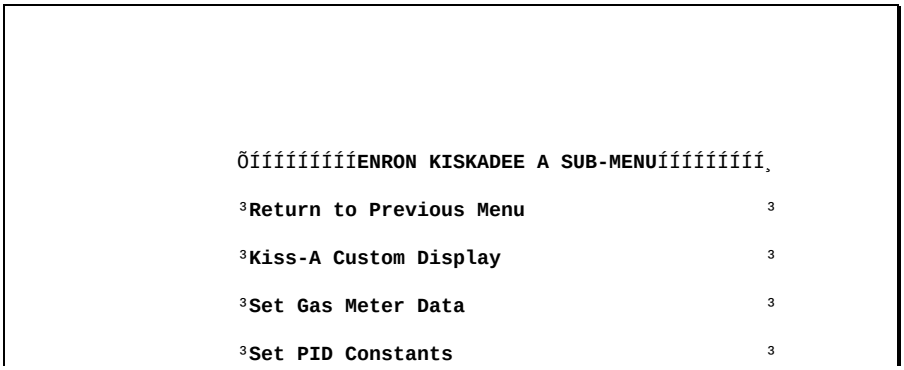
The program can be set in "Auto-Menu" mode using the command SET MENU ON. In this mode, the main program menu will automatically pop-up whenever the user is at a command prompt. This will prevent the untrained user from being left in manual command mode by accident. From the main menu the operator can select a Menu Off option that will escape to command mode as well as cancel the auto menu function.

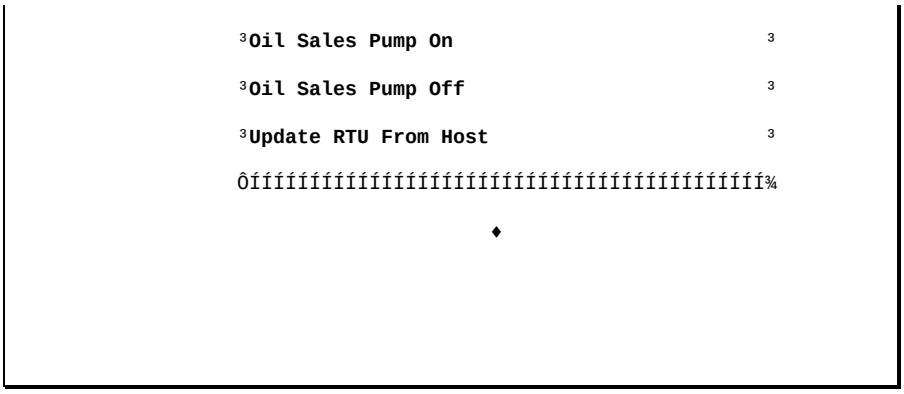
Whenever the main menu is up and a message from another task is sent to the local console, the main menu is automatically exited and the message is displayed. If Auto-Menu is off the program will remain at the command prompt. Otherwise, the main menu will be redisplayed after the message is displayed. This can happen so fast that it might appear as if the menu simply flickered and the message may not be seen at all.

If a Microsoft compatible mouse or trackball is installed on the computer, the program will automatically recognize it an make use of it for all menu and editing operations. For best performance, it is recommended that a bus type mouse be used instead of the slower serial type mouse. Also, if the SCADA program is told to use the com port already taken by a serial mouse, the mouse will not operate while the SCADA program is running. If this is a problem, delete any references to the affected com port from the DAT file that controls the overall system setup.

15.3 USER MENU SYSTEM DESIGN

The SCADA program also supports a user programmable "pop up" type menu system that is available only to the local user. This menu system, referred to as the User Menu, is used for any custom functions unique to each location. The menu allows for quick selection of preprogrammed actions by displaying a list of choices to the operator. The user can use the arrow keys of the keyboard to move up and down the menu list. If a Microsoft compatible mouse is installed on the computer the Menu system will detect it and activate its use.





The menu commands are contained in text files, called menu files, that have a default extension of ".MNU". Only one menu file is active at any one time, although other menus can be loaded at any time via the keyboard or via a menu pick. This allows for a fairly complicated system of menus to be set up with separate actions on individual menus.

The basic idea is to place a descriptive message as the menu title, such as "SHUT IN PLATFORM". This is a message that an untrained operator can use easily without knowing the details of the system operation. When a selection is made, the local task will begin processing a list of commands from the menu file that are related to that menu item. The list can be as long as desired (up to a total of 40 command lines for the entire menu), although the usual practice is to have the selection start execution of a separate command file.

NOTE: Care must be taken when setting up menu commands directly in the menu file rather than having the menu execute separate command files. The commands to look out for are the ones that include special codes that get expanded at run-time. (See the section on EXPANDABLE CODES for a list of these special codes.) Using such commands in a menu file may cause the codes to be expanded at the time they are queued up by the menu system rather than when they are processed. This may cause unpredicted and undesirable results.

Each menu item can have an optional security level assigned to it. If passwords are in use, the current operator must have a security code high enough to execute each menu item. Processing of the item will be denied if the user is not eligible.

15.4 USER MENU LOADING

The user menu system does not automatically install itself when the program is started. A MENU LOAD command must be used in order to load in an initial menu file. An optional file name can be specified to override the default "RTU.MNU" file name. If the user menu is called before a menu file loaded, then a default command "MENU LOAD" is sent to task 0.

Once loaded, the user menu system will contain both the menu prompts and the command lines. No further disk system accesses are required to display and process the basic menu items. This is important to understand because changes made to a menu file will not take affect until the user menu file is reloaded. This process can be automated by making the menu load function one of the menu pick items.

15.5 USER MENU ACTIVATION

The "pop-up" action of the user menu system can be activated by pressing SHIFT-F9 (or possibly just F9), by using the USER command or by selecting USER from the main menu. Once displayed, the menu selections can be made by using the up and down arrow keys to move the highlighted bar to the desired choice.

In mouse systems, placing the mouse cursor on the desired selection and pressing the left button will also select the item. After selecting the item, it can be activated by pressing the [ENTER] key or the left mouse button twice quickly (commonly called "double clicking").

15.6 USER MENU FILE FORMAT

The user menu files are simple text files containing both the menu prompts as well as the command lines to be executed. The first character of the line determines the function of each line in the file. There are two special characters: The vertical bar | and the accent ` character.

The accent character is used to indicate the menu title that will appear at the top of the menu display. Each menu can (and should) have its own title. For example, the line

```
`RTU USER MENU
```

can be used to generate a simple title. If none is provided the menu will use the default title "RTU / SCADA MENU". Note that all special parameters like $U or $T that are listed in the title are expanded each time the menu is displayed.

The program will automatically position the menu at the top, left corner or the local CRT screen. If a position other than this is desired the menu title line can contain two optional parameters which specify the column and row for the upper left hand corner of the menu. The special separator ~ is used to delimit the coordinates from the menu title as follows:

```
`RTU MAIN MENU ~ 4 ~ 6
```

This would place the menu at column 4, row 6. This overall size of the menu window is still determined automatically.

The vertical bar character is used to indicate the start of another menu pick item. The remainder of the line will be the text that is displayed as the menu prompt for this item. For example, the line

```
|SHUT IN PLATFORM
```

would be displayed as "SHUT IN PLATFORM". Note that the leading vertical bar is not displayed but serves only to indicate that this line is a prompt line.

An optional security code level can also be specified after the prompt text. A comma is used to separate the prompt from the security level. For example,

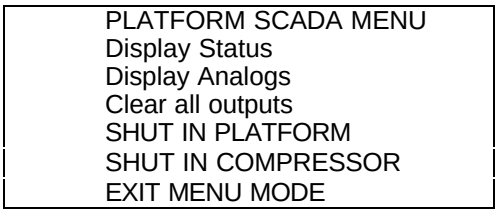
```
SHUT IN PLATFORM,3
```

will prevent a level 0, 1 or 2 operator from using this menu item by specifying a level 3 operator as a minimum requirement.

All other lines are assumed to be command lines that are associated with the most recently defined prompt line. Each menu system can hold up to 24 prompt lines and a total of 40 command lines. Consider the following sample menu file.

```
`PLATFORM SCADA MENU
|Display Status
  disp s1
|Display Analogs
  disp a1
|Clear all outputs,2
  reset outputs
|SHUT IN PLATFORM,3
  calc %1 = SHUT IN Platform
  read shutin,1
|SHUT IN COMPRESSOR
  calc %1 = SHUT IN Compressor
  read shutin,2
|EXIT MENU MODE
  force clear 0
```

This file will be read by the menu processor so that the menu which appears looks like this:



The optional parameter after each command entry is the task that is to receive the line. This allows for commands to be sent to other tasks automatically without having to use a FORCE command in each line. The default task is 0, which is the local task. Note that each menu selection can have several command lines associated with it. Normally, lengthy sequences are placed in a separate command file that is activated by the menu selection.

Note that an automatic USER command is sent to task 0 after all lines for a particular entry are processed. This has the effect of causing the menu screen to always reappear after any selection is made. The menu system can be exited by pressing the ESC key. For automatic exits, place the command "FORCE CLEAR 0" at the end of the sequence for an entry. This will cause the automatic menu command to be cleared before it can be processed, effectively halting the automatic menu pop-up. This is useful to allow the local user to monitor the progress made by other tasks that are sending messages.

15.7 NESTED USER MENUS

Because user menus can be quite complicated, a nesting procedure is available that allows "calls" to a sub-menu followed by a "return" to the previous menu. The ability to call one user menu from another user menu is capable using the following commands:

MENU READ [menuname] ; original transfer to new menu
MENU GOSUB menuname ; nested transfer to new menu
MENU RETURN ; go back to previous menu

Note that the nesting only works if the GOSUB and RETURN commands are used. The system maintains a list of menu files used and each GOSUB moves one position down the list. Each RETURN moves up one position on the list. A READ does not change position but does change current menu files. This is very similar to the method used for nesting RTU command files with the normal READ, GOSUB, and RETURN commands.

15.8 CONTROL-BREAK INTERRUPT

The local keyboard has a key labeled BREAK that generates a special action within a PC. The SCADA program traps this special key and attempts to bring the local CRT task back to the command level from wherever it is currently running. This key can be detected only when the program is writing to the screen or the printer. However, if the command BREAK ON is set from DOS then the break checking is extended to other functions such as disk reading and writing. This is mostly useful in displays or command files (that write to the screen or access a disk) that are stuck in a loop.

To use the BREAK key, the user must hold down the control key prior to pressing the BREAK key. This is because the keystroke is actually a CONTROL-BREAK, not just a break. When the software detects the break, it does the following:

1. Sends and ESC CONTROL-C sequence to the local task.
2. Stops processing of any command file in progress.
3. Beeps the local speaker as a confirmation.

16 TSP EXPRESSION EVALUATOR

The SCADA software has an expression evaluator that is tailored to RTU/SCADA applications. This evaluator allows simple math calculations using both channel data and special variables. The evaluator also allows for special functions to be performed to assist in these calculations.

The evaluator is an interpreter, meaning that is scans and decodes all lines as they are processed. This allows for simple text editing of files containing the expressions, but hinders the performance because of all the text handling involved. Applications using extensive math calculations should use at least an "AT" class computer for best results.

16.1 EVALUATION RULES

The evaluator uses a simple left to right scan to generate all temporary results. No operators have precedence over any others, so parentheses should be used to specifically determine the order of evaluation.

Normally, an expression can be entered as an algebraic equation as in:

TOTAL = TOTAL1 + TOTAL2

Here, the values TOTAL, TOTAL1, and TOTAL2 would have to be defined somehow as either channels (using channel tags) or as variables (using LOCAL or PUBLIC commands).

16.2 IF EVALUATION

The IF processing in the SCADA program uses the expression evaluator to determine if the IF is true or false. A zero result is false, and any non-0 value is considered true. The evaluator will completely calculate the expression and return the result to the IF command processor which determines if processing should continue or skip to the next ENDIF.

When using the evaluator on IF statements, it is permissible to use parentheses in the equation. This can be helpful in some of the more complex equations. A simple equation using parentheses might be

IF ((TOTAL1 + 100) < TOTAL2)

16.3 MATH OPERATORS

The evaluator uses the standard computer notation for all operators as in:

+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponents
%	Truncate to word from floating point
&	Logical AND
|	Logical OR
=	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than

>= Greater than or equal to
>> Shift right
<< Shift left

The actions of the OR and AND math operators provide a bitwise logical operation rather than a simple 0 or non-0 operation. The operators return the actual bitwise function such that 3 OR 4 returns 7 instead of 1. This allows bit mask type operations to be done with the normal logical AND/OR methods.

As mentioned, none of these operators take precedence over any others, so parentheses should be used to insure the logical grouping of terms in an equation.

Examples:

```
if (x >= y)
if (y <= x+1)
if x <> y
y = x << 2   ; shift x left 2 binary places
y = x >> 2   ; shift x right 2 binary places
```

16.4 STANDARD MATH FUNCTIONS

Standard functions are available to provide math operations not available with simple +,-,*, and /. These operators all take a single argument, which may be 0 for functions that do not really need an argument.

SQRT() Square Root
SQR() Square
INT() Integer Portion of floating point number
FRAC() Fraction part of floating point number
PI() Predefined value for PI.
ABS() Absolute value of a number.
ROUND() Rounded to nearest whole number.
TRUNC() Chop off fractional part.
SIN() Trig Functions....
COS()
TAN()
ARCTAN()

These functions are used in any place a value or variable is allowed, and they must have an argument. For example:

```
TOTAL = SQRT(VALUE1) + SQR(value2) + (2 * ABS(Value3))
```

16.5 SPECIAL SCADA FUNCTIONS

The special functions provide features not normally available in an equation solver, but are useful in the RTU/SCADA system. These functions perform some type of system operation rather than an algebraic type function. All special functions start with the "@" character and require some type of argument. The arguments for many functions are dummy arguments whose values do not matter. However, the number 0 is usually specified in situations where a dummy argument is required.

The functions below return various RTU and system information. An xx is used to indicate situations where a dummy argument should be specified. An x is used to indicate situations where a valid number must be specified.

@KEY(xx) Returns 0 for no key hit, or ASCII code for key.

@GETRTU(xx) Returns current RTU number.
@SETRTU(x) Sets the RTU to the passed number x. The argument can be between 0 and the maximum RTU number. A 1 is returned if successful, otherwise a 0 is returned.
@ONLINE(x) Returns logical online status for port x. Returns 0 if not online, 1 if online.Use -1 to default to port used by the current task.
@COMPORT(x) Returns communications port number for task x.
@COMSTAT(x) Returns physical online status for port x. Returns 0 if not online, 1 if online.
@FLAG(x) Returns value of specified flag (1-512).
@FLAG(0) Returns 1 if any flag is set, otherwise returns 0.
@RFLAG(x) Returns value of specified RTU flag (1-512).
@RFLAG(0) Returns 1 if any RTU flag is set, otherwise returns 0.
@DBERR(xx) Returns integer value for status of current database. If no error then 0. If no current database then -1.
@DBREC(0) Returns current database record number.
@ACK(x) Returns 1 if last block command for task x was ACKed, otherwise returns 0. Use -1 to default to current task.
@YES(xx) Returns 1 if most recent Pause command was answered YES or most recent exit from an entry screen was done with a save, otherwise returns 0.
@NO(xx) Returns 1 if most recent Pause command was answered NO or most recent exit from an entry screen was done with an ESC, otherwise returns 0.
@IMAGE(xx) Returns 0 if no image errors. Usually called after loading the image to see if it was properly loaded.
@IDLE(xx) Returns value of idle counter.

Examples:

```
pause, Enter Y or N, 10, N
if @yes(0)
 msg YES was entered
else
 msg NO was entered

block SCAN a1:a5 E
if @ack(-1)
 msg Last Block was successful
else msg Last Block did not get an ACK response
endif
```

The functions below return information about the system. All of these functions use a dummy argument.

@TIME(0) Total seconds since midnight (same as @DAYSECS)
@HOURS(0) Hours in current day
@MINS(0) Minutes in current hour
@SECS(0) Seconds in current minute
@DAYSECS(0) Total seconds since midnight (same as @TIME)
@DAY(0) Date number of the month
@MONTH(0) Month of the year
@YEAR(0) Four digit year value
@DAYS(0) Number of days in the current month
@FREE(0) Returns the total number of free bytes of memory
@CPU(0) Returns CPU type: 86, 186, 286, or 386 if in real mode. -286 or -386 if in protected mode. Intel 486 is detected same as 386.

The functions below return information about a channel. The argument for each function must be a channel name, channel ID, tag name, or a number that equates to a unique internal channel number. Most uses will simply pass the channel name for which the information is needed.

@LOW(id) Low setpoint value

@HIGH(id) High setpoint value
@DB(id) Deadband Setting for a channel
@WAIT(id) Seconds point waits before alarming
@DELAY(id) Seconds left before alarming
@STATUS(id) Status of a channel
1 = enabled
2 = on hold
4 = remote point
@ALM(id) Alarm state of a channel
0 = no alarm
1 = Timing
2 = New Alarm
3 = In Alarm
4 = Deadband
5 = Needs Reset

@RTU_NUM(id) RTU Number that "owns" a channel
@RTU_PT(id) RTU Relative point number
@SYS_PT(id) System relative point number
@TYPE(id) Channel Type Code
0 = Not a formal channel (variables, etc)
1 = Status In
2 = Status Out
3 = Analog in
4 = PID Calculation (and analog out)
5 = AGA3 Gas Meter
6 = Totalizer
7 = Counters
8 = Functions (rate and average)
9 = Timers
10= Values

@PTR(id) Unique Table Index pointer to internal variable
@POINTS(id) Returns number of RTU points for type of channel specified.

Examples:

```
calc x = @low(A1) ; get low setpoint for channel A1
calc y = @db(RTUBATT)   ; get deadband for channel RTUBATT
echo Month $(@month(0)) has $(@days(0)) in it
```

16.6 EXPANDABLE $ CODES

Commands can contain special codes that get expanded into numbers or text at the time the commands are processed. These codes can be helpful in creating generic command files that can be used by different tasks or different RTUs. For example, the command GOSUB $S.LIN can be placed in the startup file of any RTU to have the link settings automatically loaded. At the time the command is processed the $S is replaced by the system name as defined in the DAT file on a NAME command line.

Except for literal strings denoted by sets of double quotes, all codes begin with the dollar sign ($) character. A list of all codes and a description of each is as follows:

$S System name.
$L Current link number.
$T Current time of day.
$D Current date.
$#T Time of current database record.

$#D Date of current database record.
$.T Last update time for the current RTU.
$.D Last update date for the current RTU.
$@ Last update time and date for the current RTU.

$N Text name of the current RTU.
$R ID of the current RTU.
$$ A dollar sign.
$U User's identification from password system.
$0-9 Command file command line parameters.
$%0-9 Task string variables.
$( ) Any TSP expression.
" " Return string inside of quotes, with or without a $.

Several examples of how the expansion would work are as follows. The underlined sections of each line on the right are the result of the $ codes in the line on the left.

```
     ORIGINAL TEXT COMMAND          EXPANDED TEXT
MSG The time is $T                        The time is 11:24:45

Msg The value of v1 is $(v1) Psi          The value of v1 is 345.23 Psi

Lib load $S.lib                     Lib Load VR123.LIB

This will $%1                       This will SHUTIN COMPRESSOR

Msg You earned $$100                You earned $100

Msg Shutin $1 for $2 secs                 Shutin VR123 for 45 seconds
```

16.7 TEXT STRING EXPRESSIONS

Each RTU task has a set of string variables that can be used to store temporary text expressions. The most common use is to store a short description of an output action that can be referenced in a generic command file. The string variables can be used for any purpose where text data must be stored or passed among program files.

The string expressions are identified by a percent sign % followed by a number from 0 through 9. Therefore, each RTU task has 10 string variables associated with it that are completely separate from the string variables of other tasks. The CALC command has a special form that is used to set and manipulate string variables. They can be displayed with a special $ operator as described above.

To set a string variable, a calc command in the form of

CALC %1 = This is a message

is used. Everything after the equals sign is placed into the string variable just as it appears on the line. Several string variables can be combined by putting + signs between each one as follows:

CALC %1 = This
Calc %2 = "is a"
Calc %3 = Message
calc %5 = %1 + %2 + %3

After this command sequence runs, the value of string variable number five would be, "This is a Message." Note how quotation marks were used to combine more than one word into a single group when %2 was set. To clear out the contents of a string variable, simply use the CALC command with blanks to the right of the equals sign.

17 PROGRAM VARIABLES

17.1 PUBLIC AND LOCAL VARIABLES

During program execution each task can define variables which can not be accessed by other tasks. Variables are useful for such things as temporary calculations or prompting the operator for input and receiving numeric values. The default number of variables allowed for each task is 0. To change the number of variables that can be defined for a task the VARIABLES command must be used in the main configuration (DAT) file. The number of variables cannot be changed while the program is in operation. For more information about how to allow more variables for a task refer to section 7.4 of this manual.

Once the program is started, variables can be defined for a task by using either the LOCAL or PUBLIC commands. Once defined, public variables can be used by different command files. However, local variables can only be accessed by the files that declare them or by subsequent files called with the GOSUB command. Upon exit from a particular command file, all local variables defined within that file are automatically deleted. However, all public variables will remain defined. The only way to get rid of publicly defined variables is to explicitly delete them using the RELEASE command.

Examples:

```
LOCAL total1 total2 total3
PUBLIC new_value old_value
RELEASE varname
RELEASE ALL       ; delete all publicly defined variables
DUMP VARS         ; display nesting level, name and value of vars
```

17.2 VARIABLE ACCESS

Variables can be accessed in much the same manner as channel data. Channels are used to manage data that is relevant to the entire system. Variables are used to hold real values (floating point data) that are relevant to only a single task. RTU channels can be referenced by Tag Name or Point ID. Variables are always referenced by name.

RTU channels are accessible at any time from any task or program running on a task. Variables have what is called a "scope," which means that they are only accessible by certain portions of the system. The scope of a variable is always limited to the task that created it. Therefore, a variable named TOTAL for task 1 is completely different from a TOTAL on task 2. This is NOT TRUE for channel data, where a *channel* named TOTAL would be accessible from anywhere in the system.

Another scope limitation is a variables status as Public (global) or Local (private). A public variable is permanently declared to exist for a given task. Once defined (with a PUBLIC statement), the variable will remain until a RELEASE command frees its memory space. A variable defined with a LOCAL statement will only exist for the duration of the program that creates it. A Local variable is accessible from the creating program, or from any programs that are called from within it. Therefore, the scope of Local variables is restricted to the area where the variable was defined.

If any subprogram defines a variable with the same name, both variables are kept separated and their scopes will not overlap. For example if PROGRAM2 creates a variable named TOTAL and then does a gosub into PROGRAM3, the same variable TOTAL will be accessible from there as well. However, it PROGRAM 3 creates its own local variable named TOTAL, it will be separate and only accessible to PROGRAM3. The new TOTAL will override the existing TOTAL variable. A locally created variable will also override any existing Public variable with the same name.

Variables are generally created as local unless there is some specific reason for having a variable that

maintains its name and value across program calls. Local variables are used for temporary calculations, loop counters, and other numeric purposes that only exist for a short time. For example, a variable named NEWRATE may be public to contain the value for the new flow rate. The rate will be set by some programs that adjust an output point. Because it is public, NEWRATE will be available at all times to all programs, even those run at separate times. However, while trying to set the rate, a sub-program may need to do some looping and delaying while the outputs are being controlled. The sub-program can create some *local* variables called X, Y, and Z that are used as temporary counters. When the sub-program terminates, the memory used by X, Y, and Z is automatically released.

When an expression is being evaluated, the variable lookup system uses a specific path to locate a particular value. This is important to understand because duplicate names will be located based on these rules. If the chance for a duplicate exists, then it is best to program around this and always make sure that the program flow eliminates the chance of finding the wrong name by mistake.

The variable access rules use the concept of the current task, current RTU, and local variable to minimize the scope of all channel and variable name references. In other words, a simple variable reference is assumed to be as close to the current point in the system as possible. The lookup system uses a specific order and stops looking as soon as a match is made. The sequence of lookup for all variable references is as follows:

1. Check current RTU for a tag name (like PT110) match in the channel order of Status, Output, Analog, PID, Meter, Totalizer, Counter, Function, Timer, and Value.

2. Check current RTU for a point ID (like A10) in the same order as the tag name search above.

3. Check local task variables working backwards from the current sub-program to any higher level programs.

4. Check public task variables.

5. Check current database tag list.

Note that certain scope changes can be made through the use of overrides embedded in the variable name. For example, it is possible to reference an RTU channel from an RTU that is not current by appending the RTU name ahead of the variable name with a single dot. The reference RTU2.S10 will cause the lookup system to go directly to a unit named RTU2 and check for a tag or point reference of S10. Without the RTU2 prefix, the lookup would have occurred in the current RTU.

Database references can also be directly accessed by appending a pound sign to the end of a variable reference. This will speed access of these variables in large systems because they would otherwise be the last place to be searched. This also works around problems with database fields that are named after actual channel names. See the separate Database Reference for more information.

17.3 TEXT STRING VARIABLES

Each RTU task has 10 text string variables, %0 - %9, which provide basic support for string (text) data. These variables can be set by using the LET, CALC, or INPUT commands. These strings can be referenced in displays and messages with the $% operator. For example,

```
CALC %1 = THIS IS A TEST
```

will set the variable %1 to be equal to the whole string "THIS IS A TEST". The command

```
MSG $%1
```

will cause the string THIS IS A TEST to be displayed. When a variable is preceded by the $, that instance of the variable is replaced by the actual contents of the variable.

17.4 FLAG BIT VARIABLES

SCADAWARE's Flag bits are used in programming operations to hold an arbitrary off/on status that can be easily set, cleared, and tested. A Flag is similar to a Status Input in that it holds and off/on value. Flags, however, do not have all of the channel attributes and processing overhead associated with normal data channels. "Setting a flag" allows one part of a program to pass control information to other parts that can check and see if a specific operation needs to be performed.

For example, a menu selection may cause a flag to be set to indicate that a certain output operation is required the next time an RTU is polled. During the polling process, the task connected to the remote RTU can check the flag and see if the operation is required on this cycle. Flags are very similar to control bits (outputs) in a PLC system.

There are two types of flags in the SCADA program. One type (System Flags) is global to the entire program, while another type (RTU Flag) is local to each separate RTU. The System flags are used to control system wide functions, while the RTU specific flags are used for operations unique to that unit. The actual use for most of the flags is completely up to the system designer, although a few conventions have evolved which will be mentioned later.

Flags are much like boolean variables which can only be set to off or on, 0 or 1. Flags do not have names like variables, but are accessed by their numeric position in an array. Flags are set OFF or ON by using the FLAG command for System flags, and RFLAG command for RTU Flags. The status of a single System flag can be checked by using the @FLAG(x) function in an expression. RTU Flags are similarily checked with @RFLAG(x).

The global System flags exist in a predefined array of 512 off-on variables that can be accessed at any time by any task. Flags 1-496 are available for general purpose use. *The 16 System flags at 497-512 have predefined purposes* which correspond to various SET parameters as explained below.

512	SET RFILE
511	SET DELAY
510	SET EJECT
509	SET BACKUP
508	SET WARBLE
507	Warbling Status
506	SET TICK
497-505	Reserved

These special flag positions are not affected by any FLAG operation other than the function @FLAG. This allows the status of these parameters to be tested for programming purposes. However, the off/on status of these flag bits can only be changed with the corresponding SET function.

A special test is available to determine if *any* system flags are set. This test can speed up certain operations where a large number of flags must be individually tested. Using the special test at the beginning of a procedure will allow a fast exit if none of the flags have been set. The flag number to test is 0, which is not a real flag, but only a signal to the evaluator to see if any flags at all are set. The following program stub demonstrates this technique.

```
if @flag(0)
  return          ; nothing set at all so quit right away
else              ; at least one flag is set so test them
  if @flag(100)
      (do something related to flag 100)
  endif
  if @flag(200)
```

```
      (do something related to flag 200)
  endif
endif
```

Note that @FLAG(0) tests only flags 1-496 for non-zero. The upper 16 flags which are used for system-wide settings are not tested with this expression. They are also not affected by the FLAG RESET command which is used to clear all system flags.

The RTU relative flags are nearly identical to the system-wide flags. The RTU flags are set with RFLAG (note the R in front of FLAG) command. The status of the RTU flags is checked with the @RFLAG(x) function. The FLAG and RFLAG commands are identical except that they affect a different set of flag variables. They use command line options that determine what happens to the flag(s) that are referenced immediately ahead of the option. The options are:

OPTION	ACTION
ON, True	Set the flag On
OFF, False	Set the flag OFF
FLIP	Reverse the current state of the flag
CLEAR	Set ALL flags to 0 (except System Flags 497-512)

FLAG and RFLAG can reference one single flag or a group of flags in a single command. This is done with a notation very similar to a group Channel reference. The flag range is specified with either a colon (:) or a double dot (..) notation between the end flags of the range. For example, FLAG 30:55 will reference flags 30 through 55 as a single group.

More than one group can be referenced on the same line. This makes it easier to program multiple flag operations because they will occur at the same instant and the program is easier to read. The following line is valid:

FLAG 1 ON 10:20 OFF 101:109 FLIP 32 ON

The following program stubs will demonstrate a typical use of flag variables to control the pulsing of outputs at a remote RTU from a Host RTU where this program would run.

```
;  Program to set a flag and cause call to RTU for output control
cls
cursor 1,5 msg This action will call the RTU and pulse an output
cursor 1,10
pause,Do you want to continue,N,45
; get here if they answer Y above
sele RTU1
RFLAG 5 on  ; Set flag 5 for output and timer number 5
```

The above program stub would request confirmation by the operator and set flag number 5 for this RTU. Another task will process some sort of link or callout file that will test the flag to see if an output pulse is required.

```
; This proc is run by TASK 1 at the time a link is made
set online on
if @rflag(5)  ; see if output pulse required
  block calc T5 = 30   ; pulse the remote output
  rflag(5) off          ; clear the local flag
  sleep 10        ; wait for output to take affect
endif
block read download   ; proceed with download from RTU
```

This second procedure is the one that would run when the RTU is being polled. Before requesting the download from the RTU, the program will check the status of RFLAG number 5 for a TRUE or FALSE condition. If it is TRUE, then a command is sent to the other unit to pulse an output. Note that after the pulse command is sent the RFLAG is cleared (set to off) so that the action will not occur until the flag is again set at a later time.

18 PASSWORD SYSTEM OPERATION

18.1 PASSWORD SYSTEM DESIGN

The RTU/SCADA program has a password security system that is designed to allow flexible control over who can perform certain operations on the system. If enabled, the passwords require each user of the system to "LOGON" using a private password assigned by the system supervisor. The passwords are encrypted to make it difficult for a "hacker" to get into the system by looking at the password storage file.

Each user that is set up in the password has four parameters:

1. A unique password.
2. A 6 character ID code (such as Joe, Dave C.).
3. A longer test description (such as John H. Doe).
4. A security level.

The ID code is available as a system parameter with the $U code. This allows the current user's identity to be used in messages. See section 16.6 for further information on expandable codes.

18.2 SECURITY LEVELS

The normal security levels inherent in the program are 0,1,2, and 3. These represent the following capabilities:

0 - Any user, even one who is not logged on.
1 - A basic user who will do display functions only.
2 - An operator level user who can control outputs, etc.
3 - A supervisor level who can do any program function.

Each command has a specific security level that is required for an operator to process the command. These levels are listed in the command reference for each command. Other types of execution, such as command file processing, are exempt from security level checks. These checks only affect "human" operators.

It is possible to assign levels higher than 3 to command files and menu entries if desired. This may be used to prevent operation of menu picks or command files that use the SET LEVEL option. As long as a user's code is equal to or greater than that required, the command or function will be processed.

For example, a command file can be set up that requires a level 10 user to operate. The password system can contain a "super-user" who is the only one capable of executing this file. This may be used for major control functions such as facility shut-ins where only specific operators are allowed to perform the desired action.

18.3 PASSWORD SECURITY RELIABILITY

Although this password system is more than adequate for most applications, it is not fool-proof. Caution should be used when implementing complex operations that are to be protected with the system. Contact TEST for specific information on implementing higher level security if required for your applications.

The security system will prevent unauthorized use of certain commands and menu operations. However, the security system will not prevent a malicious user from disrupting the overall operations of the computer. Passwords should only be used to prevent accidental or inadvertent operation of functions and to control access for supervisory reasons. Passwords should not be used to prevent dangerous activity of any kind, and cannot be relied upon to prevent knowledgeable users from accessing key components of the system.

A simplified approach to password protection has been used at many installations that bypasses the complex internal password system. This is presented in the next section as an alternate to the use of the full password security system.

18.4 SIMPLIFIED PASSWORD PROTECTION

Most users do not want to be bothered with passwords for the vast majority of their operations. It is simply not convenient to log on and off and keep track of the security codes for typical SCADA system applications. However, there is the occasional need to prevent inadvertent or unauthorized use of the system for specific functions. This is desirable in drastic actions such as facility shutdown or restart commands. It is fairly easy to implement a simplified password system that will prevent the casual user from accessing protected functions. This technique is presented below.

The simplified security system is based on using text string variables to hold the "secret code" and then use a simple comparison to see if the proper code has been entered. This simple procedure can be called before all sensitive operations, and will halt execution of the TSP program if the incorrect entry is made. This is easy to implement and manage, but is also fairly easy to work around if someone familiar with the system really wants to . Of course, someone with that capability will probably find other ways to "beat the system" with any security plan.

The following program will ask the user to enter a password, check it, and set a flag if the proper entry was made.

```
; Password check program called CHECKIT
flag 100 OFF            ; Flag 100 control Password protected code
cursor 1,10
calc %1=""        ; clear out any existing text in string no. 1
input, Enter password, %1
if %1 = `Password'   ; change this line to change password
  flag 100 ON
endif
return     ; go back with flag 100 set properly
```

This CHECKIT program can be called from any other program that wants password protection. The user's program will gosub to CHECKIT, and then test FLAG 100 on return to see if the proper password was entered. The following program will pulse a timer if the proper password is entered.

```
; Pulse timer with password protection
cls
msg This procedure will pulse an output timer
gosub checkit    ; run the password check routine
cursor 1,12
if @flag(100)    ; flag is set if password OK
  msg Password is correct.  Pulsing Timer 10 now
  calc T10 = 30
else
  msg Password INCORRECT.  Access denied.
endif
return   ; end of program
```

When this program is run, it will present a message and then gosub to the password check. If the proper password is entered, FLAG 100 will be set and the program will proceed with the pulse action. If it is incorrect, a message will be displayed informing the user that it was wrong and that the action will not occur.

This is a simple example, but the principle is the same for even the most complex protection scheme. It is suggested as a workable solution for protecting certain operations while keeping the overall system as simple as possible.

19 PID CONTROL OPTION

19.1 INTRODUCTION

This section describes the Proportional-Integral-Derivative (PID) control option in the TEST RTU/SCADA system. The control algorithm is a classic PID loop with several enhancements suitable for industrial and oilfield applications. Both open and closed loop methods are supported. This document is not an in-depth discussion of PID loops. It provides information on how the PID control option is implemented in the system and offers tips and suggestions for its use.

The PID loop is provided as a new channel type derived from the Value channel. It has all of the normal Value channel attributes such as alarm mode, units, and decimal palaces. The PID type adds in the features needed for the control algorithm, and these are covered below.

19.2 PID BASICS

PID control is a classical but complex method of maintaining control of a dynamic system. The PID control system monitors a process signal (measured variable or MV) and compares its value to a setpoint (SP). The output of the PID will control another variable that will in some way be related to the measured variable. The typical example used to explain PID systems is a hot water heater. The measured variable will be the water temperature. The setpoint will be the desired temperature. And the output will control the amount of heat added to the water tank.

Note that the input and setpoint variables are in the same units, such as Deg-F. The output is also in the same units, although actual practice avoids thinking of the output in terms of normal values. Instead, the output is normally thought of in terms of percent full scale for the output. The PID calculates an output value that would be "added" to the input value to make it equal the desired setpoint. In our example, we cannot add degrees. We can add heat, so there must be a relationship between added degrees and heat input. This is where the complexity comes in.

Most real-life systems such as a flow valve or a water heater have many interrelated variables involved. If our example is simple, we can adjust the heat input to match the heat output. A given heat input will cause the temperature to remain stable as long as nothing changes. This is not normally the case. Something as simple as a water heater can have many dynamic conditions such as:

1. How much hot water is being used at the moment?
2. How fast is the water usage changing?
3. How much water is in the tank?
4. How good is the gas being used to heat the water?
5. How much heat is being lost through the tank wall?
6. What is the current desired temperature?

These sorts of variables are common in any control system. The water tank is used here because its easy to understand. The point of the PID algorithm is to provide optimum control for a dynamic system having changes such as those listed above. This is done by monitoring the input signal according to a complex formula, which we will not cover in detail here. The end result is what is important, and this can be understood without ever looking at a complicated math equation. But for those who are really interested, the TEST PID calculation uses a common digital 'position method' based on the following equation:

$$m(i) = [Kc * e(i)] + [T * Ki * \sum_{k=0}^{i} e(k)] + [(Kd/T) * (e(i) - e(i-1))]$$

Where:

T	Sampling interval
e(i)	error at ith sampling intercal= S(t) -X(t)

e(i-1) error at previous sampling interval
m(i) controller output deviation (PID output)
Kc Proportional Gain
Ki Integral action time
Kd Derivative action time

The P, I, and D in the PID term do not mean "Piping and Instrument Drawings." The letters stand for Proportional, Integral, and Derivative. These terms refer to the three components of the complex equation that calculates the output of the PID loop. Understanding how each one relates to the real world is the key to understanding the PID system. In quick terms, each factor relates to the following:

Proportional - How far is the input from the setpoint?
Integral - How long have we been off the setpoint?
Derivative - How quickly are we approaching or departing from the setpoint?

Note that alternate terms are often used for Integral and Derivative. The Integral term is also called RESET because it attempts to restore the input to its setpoint. Derivative is often called RATE because it relates to the rate of change of the input. This document will stick to the PID terms because they are the classic names found in most control material.

19.3 P - PROPORTIONAL TERM

Understanding proportional control is the key to understanding the entire PID system. The Proportional output is directly related to the difference between the input and the setpoint. If the input is 32, and the setpoint is 40, then the proportional output is based on the value 8 (40-32). The difference is what is important, not the absolute value of the variables. The proportional output is given a factor called GAIN that is simply a multiple of the calculated difference. If the gain is one, then our difference of 8 will produce an output of 8. However, if the gain is only 0.5, the difference of 8 will produce an output of 4 (one half of 8).

TEST's system always uses gain as the method of expressing the proportional term. Some systems use the inverse of gain times 100, called Proportional band, but this is confusing to most people. This document will always refer to the proportional term as a number that represents the multiplier for the difference error. A higher gain will produce a higher output for a given difference from the setpoint. Many systems get by fine with a gain between 0.1 and 1.0, although any gain number is possible. It all depends on how fast the system can react to changes in control.

The proportional output forms the basis of the other two output controls. Everything is based on the current difference between the input and the setpoint. If there is no error, then there is no output. This is a problem for many real world systems because a zero output will not normally produce the desired input variable. Consider the cruise control in a car. If we are at the desired speed, then the control cannot tell the accelerator to turn off all the gas. There is no error, but we still need some gas. That is the basic problem with proportional control. In order to produce an output, the input must be incorrect.

Simple control systems often live with this error and get by just fine. Your home air conditioner is a good example. You set your thermostat to produce a comfortable temperature. When it is reached, the unit shuts off. The temperature changes in a matter of minutes. When enough error is sensed by the thermostat, it kicks the unit back on. The temperature adjusts, and the cycle completes. But note that no temperature control occurs unless there is an error.

If you dial in 74 degrees you would expect to get a room that was maintained at exactly 74 degrees. This is not usually the case because the temperature is sensed at only one point in the room. A temperature of 74 at the thermostat may mean that its 81 degrees were you are sitting. If you are too warm, you will adjust the thermostat downward so that the resulting temperature at the desired point is 74 degrees. This may require a thermostat setting of 68, and that is where the manually entered error comes in. This setup will work fine once the magic setting is found and nothing changes. But different weather or more people in the room will change the

dynamic conditions such that the magic number no longer works.

Room temperature is not that critical, and we don't mind adjusting the thermostat every now and then. Industrial control systems are usually more demanding, and we need a way to have the system automatically adjust to load changes. The other two terms of the PID loop take care of this by further manipulating the normal proportional output to maintain the desired setpoint under changing conditions.

19.4 I - INTEGRAL or RATE TERM

The second term, Integral, monitors the time that an input is off the setpoint. It then adds or subtracts from the proportional error to cause additional (or less) corrective action. The Integral term does not act alone. It only alters the effects of the proportional term. Proportional was specified in gain, which has no units. A gain of 1.2 is simply 1.2 times the difference. Integral is specified in units of minutes. A value of one means that departure from the setpoint for one minute will result in one unit of proportional correction being added to the current output.

The value given to the integral term tells the PID loop how many equivalent proportional corrections to make for every minute that the input is off the setpoint. This is a simplification, but is close enough for now. So, a steady state system that is off the setpoint will have its PID output change every minute by an additional value equal to the proportional output for the same setpoint-input difference. A larger value will cause a higher change to the output for a given amount of time away from the setpoint. Note that the integral term is additive, and produces an ever increasing (or decreasing) error correction as long as the input remains off of the setpoint.

Integral is helpful in situations where the load on a system changes from time to time. In our water heater example, the desired temperature will be maintained by proportional alone as long as the amount of water flow does not change. We could easily balance the heat-in with the heat-out by adjusting the desired temperature until we found an error that did the trick. But we would always be off the setpoint, and would have to get the desired temperature by artificially raising the setpoint to generate the desired error.

The integral term will keep an eye on the input value and make adjustments to the output based on time as well as error. Eventually, the output will be sufficiently adjusted to cause the desired effect on the system. The setpoint will be reached, but the output will not be zero. The output will be the value necessary to compensate for the load on the system at that moment. Changes in the load will be detected over a period of time, and the system will balance.

19.5 D - DERIVATIVE or RATE TERM

The third term in PID is derivative, or rate. It monitors changes in the input to see how fast or slow it is changing relative to the setpoint. Slow moving changes do not require much compensation in order to keep the system stable. But a fast moving input will require anticipation on the part of the PID loop in order to keep things from getting too far off too quickly. Derivative is the least often used term because most systems have a fairly consistent rate characteristic. But small values for the Integral term can often be used to cause an improved response in systems that can get sluggish if the input strays too far from the setpoint.

The Derivative term is also specified in minutes. It tells the PID calculation how many minutes it will take to reach a certain level of output in advance of the time in which the proportional output would have done the same thing. In effect, the derivative term anticipates the required output and jumps ahead of the proportional value.

19.6 NORMAL AND REVERSE ACTING CONTROLLERS

The PID calculation uses a positive error equal to the Setpoint Minus Input. An input that is lower than the setpoint will generate a positive error. A higher input will generate a negative error. The normal thinking

process tells us that if an input is lower than the setpoint we will increase an output to make the correction. If the temperature is to low we want to add more heat to the tank. Too slow, then add speed. It makes sense and is easily understood. This is referred to as 'Normal Action' and is intuitive.

Some real systems, however, require the opposite action to occur. For example, a pressure that is too low may be increased by closing a valve. This means that a higher error will require a lower output, or a 'reverse action' from the control system. This may be done by the final control element (such as the control valve) or it may have to be built directly into the PID control system. TEST's system can easily provide reverse action if required by using a negative value for the Proportional Term. This effectively reverses all of the PID calculations causing a higher error to generate a lower output. The I and D terms are specified in minutes, however, and are always positive numbers. Their action will follow that of the P term to provide the reverse action as well.

19.7 PID TERM CHANNELS

The starting point for all PID control is the setpoint value. This can be any type of SCADA channel type such as Analog input or Value channel. The PID calculator will simply get the current value of the specified channel on each pass through the calculation and use that value as the desired setpoint. Where the number comes from is not important. The number enters the PID calculation in real units such as PSI or DEG-F.

The next thing normally needed for the PID loop is the input variable. Like the setpoint, this can be any type of SCADA channel. The PID calculator gets the input channel's value and compares it with the current setpoint. The input also has real units and they should be the same as the setpoint channel.

The calculated output of the PID loop is also in the same real units. The output is the adjustment to the input that will move it towards the setpoint based on the recent history of the inputs movement. However, real life systems cannot normally add to the process in terms of the input and setpoint variable. If we are monitoring pressure, we cannot simply add PSI. We can open a valve to increase pressure, which results in increased pressure. This is an important point. The input is the measured variable, and the output manipulates the controlled variable. They are not the same thing, but are related by some physical system.

Most mechanical or electronic PID systems operate strictly in terms of 0-100% of some arbitrary scale. For example, a pressure system that ranges from 100 to 500 pounds will have a zero point of 100 and a full scale of 500. The full range spans 400 pounds. But it is convenient to think of the input and setpoint in real terms, not in percent of full scale. The TEST PID system takes care of the conversions for you so that you can always think of your input and setpoint in the units in which they are measured by the SCADA system.

It does this by requiring that all PID loops be scaled with a zero and span value. The PID calculator will take care of any 0-100% type calculations required for various parts of the calculation.

A default option exists for each PID loop that allows some parameters to be specified in terms of percent full scale. When this is selected, the output will be generated as a percent number using the zero and span for the channel. This is the traditional PID controller method where the output signal is thought of as a percent of full scale (4-20ma, 3-15psi). The output value was internally calculated in terms of the input and setpoint units, but converted automatically into percent for convenience. This percent technique also works for some of the other output related parameters to be discussed shortly.

It is not necessary to use all three terms in the PID system be used in each installation. The P term is always needed because it forms the basis of the other two. A SCADA channel that determines the Proportional factor is always required. In many practical control loops the I and D factors can often be skipped. With this in mind, TEST's PID setup will allow the I and D terms to be ignored by simply blanking out the I and D channel names during the PID channel setup.

Some loops do not even require an input channel. This effectively bypasses the PID calculation but provides a convenient method of providing a scaled analog output from the SCADA system. If no input channel is provided in the PID setup, the PID loop will simply set the output equal to the setpoint. A change in the output can be easily done by changing the value of the setpoint channel. This type of system is covered in a separate

section a little later in this document.

```
ÕÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¸
³KISSA              PID CHANNEL        1   Of 2              REMOTE
      ³
ÔÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¾


Channel Name           Analog PID             Input Channel   P1

Tag                    P1                     Setpoint Chan   P1S

Alarm Delay Secs       0                      Proportnl Chan  P1G

Alarm Type Channel?    N                      Integral Chan   P1I

Blow Horn?  N      Play Sound? N              Derivative Chan P1D

Reset After ACK?       N                      Stdy State Chan P1S

Call on Abnormal?      N                      Use Percent?    Y

Call on Reset?         N                      Low Clamp       0

Log Abnormals?         N                ♦      High Clamp      100

Execute RTU file?      N                      Chng/Min Clamp  50

Skip Reports?          N                      Sample Period   1

Callout Group          1                      Filter Value    0

Current Value       48.9 Pct       (Units)    Offset Output?  N

Decimal Places         1                      Value at 0%     0

Input Control Channel                         Value at 100%   100

Lo Alm Val 0             Hi Alm Val 100       Min % Out Chng  0

Dead Band              0


ÕÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¸
³  <F2> Accept • <PgUp> Previous Channel • <PgDn> Next Channel • <Esc>
Cancel  ³
ÔÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍÍÍ¾
```

The final channel specification is for a steady state offset value that can be used to improve the response of the PID output. This forms a starting point from which the PID logic will start its correction. It is provided to the PID calculation from any channel just like the input and setpoint channels. The steady state channel is optional, and the value defaults to zero if no channel is provided.

The steady state value would be the anticipated value of the output for the specified setpoint. For example, it may be known that a 45% output is needed to maintain the setpoint under normal conditions. By providing the steady state number to the PID logic, it will start at 45% and begin correcting from there. This prevents startup swings that occur if the output starts at 0. The PID loop will correct itself if the steady state value is not provided or if it is within reason. But using the variable will improve the overall action of the PID by providing a built-in output offset that gets the output near the correct value as quickly as possible.

If the PID channel is using the percent method, then the steady state variable is expressed in percent full scale. If not in percent mode, then the steady state value is in the same units as the input and setpoint channels.

19.8 INPUT FILTERING

Noise in electronic systems refers to unwanted variations in a signal due to outside influences. Noisy inputs are a problem in any digital control system that takes samples at specific intervals. The noise may come from a number of sources such as interference, process variations, and electronic sampling error. The PID channel has a noise elimination factor that tends to smooth out periodic noise using a standard factoring solution. The noise factor, which is always a real number between 0.0 and 1.0, affects the sampled input according to the following formula:

Filtered Input = [(1-factor)* Input] + [factor * (previous filtered value)]

A filter value of 0.0 effectively eliminates any input filtering. A value of 1.0 will cause the highest amount of filtering.

The proper filter factor is completely dependent on the many variables in each installation. It is suggested that a value of 0.0 be used unless special considerations require otherwise. Keep in mind that the SCADA Function channels can provide input averaging which may be a more suitable means of eliminating periodic variations in an input signal.

19.9 PID OUTPUT CONTROLS

There are several internal settings for each PID loop that control the range of the output value. These can be expressed in either percent full scale or in the units of the input and setpoint. Unlike the parameters discussed above, they are not specified as separate SCADA channels. They are setup values entered directly into the PID channel configuration. These values determine the limits on the output signal to prevent huge plus or minus errors from occurring over a period of time. For example, a system that is off line may have an input value that is very far from the desired setpoint. If left unchecked, the integral term will accumulate huge output errors in an attempt to bring the input back in line. When the system is restarted, the input may correct itself quickly but will overshoot because of the accumulated output error. This is called "reset windup", and can be limited by two parameters in the PID channel setup.

Each channel has an output high and low clamp which prevents the output from going above or below the specified value. If the PID calculation determines that the new output would be beyond the specified range, it simply keeps the output value at the limit. For example, it may be reasonable to set a valve position to operate between 20 and 80 percent of full scale. The PID calculation will stop at a low of 20% or a high of 80% regardless of the results of the PID algorithm.

A rate clamp is also provided to prevent drastic changes due to step changes in the input or setpoint variables. The rate clamp is expressed in either percent full scale or in terms of the input variable and represents the largest change that will be accepted, measured in units per minute.

19.10 OUTPUT OFFSET

A setup parameter in each PID channel determines if the resulting output requires a 20% offset. Many industrial control systems use physical signals that range from 20% to 100% of a specified scale. Examples are 1-5VDC, 4-20Ma, and 3-15PSI. The intent is to avoid operation in the lower end of the scale where the mechanical or electronic equipment is less accurate.

The PID channel calculates a 'raw' binary number that will eventually be sent to an analog output board by a hardware driver in the SCADA program. The binary value will be converted into an electrical signal that corresponds to a point within an overall output range, such as 1-5VDC. How the binary number is generated is determined by the scale of the channel and the need for an output offset.

If no offset is required, than the resulting raw binary number is a reflection of the PID output in terms of 0-100% of the scale of the channel. A 35% output will result in a raw number that is 35% of the way between 0 and 32,767. The 35% value would go to a hardware board that is expected to produce an electrical signal that is 35% full scale. If the full scale is 1-5VDC, then the desired output would be 35% of the way between 1 and 5 volts, or 2.40 volts.

Some analog output hardware requires that the driving software generate the necessary 20% output offset. On those devices, 0-100% full scale corresponds to 0-5VDC, not 1-5VDC. The initial one volt must be provided by the software. Note that the board may generate 0-20ma or 4-20ma in a similar manner. The important point is whether the hardware provides the offset or not.

The PID channel allows the program to provide the correction needed to work with standard 20% offset channels. This is done by selecting 'Y' on the configuration screen at the "OFFSET OUTPUT?" field. When this option is selected, the program will automatically rescale the output to range between 20 and 100% of the full 16 bit range of the raw output value.

The correct setting for this option is dependent on the actual hardware used in the analog output system. An incorrect configuration will result in either no offset being made or a double offset being made.

19.11 MINIMIZING OUTPUT CHANGES

Some analog devices driven by a PID output require electrical power in order to change their position. Numerous small changes do not greatly affect the analog value, and waste power when the PID "hunts" back and forth as the algorithm calculates small changes in the output value. SCADAWARE provides a percent output change value that, if non zero, tells the output driver to hold an existing PID output value unless a certain percent change occurs. This value is expressed as a percentage of the current output, not full scale, so that its easy to specify what degree of error is tolerable.

An example is an electrically driven actuator that uses the analog output to position a valve or pressure regulator. Each movement of the valve requires electrical power, while a non-changing position reduces power to a standby mode. A particular application may require only a 5% precision in the valve position. In this case, a value of 5% for the output percentage change will cause the PID output to remain fixed unless the calculated output varies by more than 5%. If it does, the PID will take on the new value which will become the basis for future adjustments.

19.12 SIMPLIFIED ANALOG OUTPUTS

It is not necessary to use the complete PID system in order to get an adjustable analog output. Three alternate methods are available that will allow direct manipulation of the analog output without the use of a normal PID input channel. These methods are:

1. Use an analog input channel to drive the Analog output.

2. Use the PID channel output as its own input.

3. Use no input channel at all.

The first method will bypass the PID system completely. A PID channel is not even used. The analog output is configured to simply copy whatever is coming in on a specified analog input point. The raw value of the analog input is copied to the raw analog output without any conversion or manipulation. Thus may be sufficient in

cases where the analog input and hardware scale match the analog output exactly. This method requires that the particular analog output board be configured to map specific analog inputs into the desired analog outputs.

If there is no analog input channel available, a simplified PID can be set up that does not use a normal PID input to drive the output. The most common method will be to use the PID channel as its own input. In effect, the output of the PID is looped back into itself. This provides the functions of an intelligent analog output that will continuously adjust itself to match the value of the setpoint channel. The normal PID constraints will apply such that a change in the setpoint will cause a PID type response to the output. The output will always eventually match the setpoint, but it will move gradually according to the values in the PID terms. This prevents step changes in the setpoint from causing radical changes in the output.

When using the PID channel in this way, it is necessary to set up the scaling and steady state channels to cause the desired result. It is also necessary to assign the same channel to both the setpoint and the steady state output. With this setup, the PID output will constantly seek to reduce the error between itself and the setpoint. It will eventually attain the steady state value specified by the setpoint when the error is zero. The best way to do this is to use a scale of 0-100% for the PID range. Other arrangements can be done, but will require proper adjustment of the various parameters to insure that the PID loop will cancel itself out.

An even easier method of obtaining a simple analog output is to assign no input channel at all. This special configuration is recognized by the PID driver. It bypasses the normal PID calculation and simply outputs the necessary value to match the setpoint. No PID smoothing is done. The only manipulations are related to output scaling. This is helpful in passing an analog input to an output when the input and output are not in the same scale range. In effect, the PID channel simply rescales the input into the output without any of the normal PID considerations.

19.13 PID CALCULATION TIMING

The PID calculation is a processor intensive process similar to the AGA3 meter calculation. The effects on the computer's performance will depend greatly on the existing load on the computer and is difficult to predict. The PID channel has a setting which determines how often the PID output is updated. It is specified in seconds, and is designed to allow staggering of PID calculations in low power systems having multiple PID loops. Faster systems (286 and 386) can normally use the default value of 1 so that the PID is updated each second. Heavily loaded systems can use other numbers such that each channel will be eligible for calculation at a different time. For example one channel may be set to 3 and another to 4. In this case, they would both require calculation at the same time every 12 seconds. The other 11 seconds will see either none or a single channel calculation requirement. This spreads the load at the expense of a slower response on the PID output.

The PID speed of response is not a problem in almost all applications, so use the update setting to reduce wasted processor time. Considering that many large DCS and process control systems calculate PID loops every minute, the three or four second wait in the TEST PC based system is hardly a problem.

19.14 PID LOOP TUNING

'Tuning the loop' refers to making the necessary adjustments in the PID terms to provide optimal response in the control system. The technical details of this process are impossible to determine in a general way because each installation has its own set of physical constraints. Entire volumes have been written on the tuning techniques for particular types of equipment, so a few tips are all that can be given here. A simple method suitable for most industrial system is as follows:

. Set the I and D terms to 0 for the initial testing. Set the P term to a value between 0.1 and 1.0 to test the response of the system.

. Change the setpoint by 20% and monitor the reaction of the system. If it over reacts, lower the P term. If it is to slow, increase the P term.

. Duplicate step 2 for a low, medium, and high setpoint. Record the final output value for each setpoint.

Determine how much variation in the P term is needed to get a suitable response at each setpoint.

. Begin adding some I term, starting at a value of 0.05. Repeat steps 2 and make adjustments to the I term this time.

. If step changes in the load are anticipated, experiment with very small D terms to allow anticipation by the control system. Normally, very small values are all that is needed in industrial systems. Values from 0.05 to 0.25 are common.

. Test the final configuration at various setpoints. Keep in mind that it may be desirable to manipulate the P, I, and D values for different setpoints or operating conditions. Because these numbers are contained in standard SCADA channels, they can be easily modified locally or remotely just like any other channel.

19.15 MANUAL OUTPUT CONTROL

Normally the PID output is calculated once every so many seconds as specified in the setup for each channel. Any attempt to set the PID output value directly (by a CALC or DATA statement) will not work. This is because the value stuffed into the channel will be overwritten as soon as the PID is calculated again. In order to directly control the output, the PID channel calculation must be temporarily disabled. This is done with the TSP HOLD command. A channel on hold retains all of its normal capabilities except that it no longer does any internal conversions. In the case of the PID, the calculation is suspended until the channel is reactivated with an UNHOLD command.

This is analogous to 'manual mode' on a stand-alone PID controller. When the PID loop is on Hold, any normal channel set method can be used to manipulate the PID output. For example, a simple CALC P1=40 line could be used to set PID channel number 1 to 40. The value of 40 would mean 40% if the channel is in percent mode. Otherwise, it would mean 40 in terms of the units defined for that channel.

A simple program can be prepared to allow manual manipulation of any PID channel. The example below allows local keyboard control of any PID output with the keypad 0 and 9 keys. A direct setting can also be done by hitting the plus key and then providing the desired output value.

```
; Program SETPID. Manually adjusts PID output directly with
; local keyboard control.
cls
local x,y,z
msg MANUAL PID OUTPUT CONTROL
cursor 1,5
input, Enter PID (as in P1) ,%1
calc y = 5 ; Assume 5% change per bump
input, Enter value for adjustment,y
cursor 1,24
msg Use 1 for increase and 0 for decrease. ESC to quit
hold $%1
cursor off

:LOOP
cursor 70,1,$T
cursor 1,10 ; put up the PID text line
calc $%1=

x = @key(0)
if x =0
 goto loop
```

```
endif
; A key was pressed
cursor 1,14
if x =57  ; ASCII 9  Raise output
 calc $%1 = $%1 + Y
 goto loop
endif

if x =48  ; ASCII 0  Lower output
 calc $%1 = $%1 - Y
 goto loop
endif

if x =43  ; ASCII +  Manual Setting
 cursor on
 gotoxy 1,22
 input,Enter PID Output, z
 calc $%1 = z
 cursor 1,22
 cursor on
 goto loop
endif

if x =27  ; ASCII ESC
 cls
 cursor 1,10
 msg PID manual control of $%1 is over.
 msg Return the channel operation with the UNHOLD $%1 command
 cursor 1,24
 cursor on  return ; get out of here
endif

goto loop
; -------------- End of Program
```

This TSP procedure is an example of how the output could be manually controlled. Note that the channel is placed on hold by the procedure prior to the manual manipulation. In this example, the channel remains on hold at the end of the program. These types of features are completely up to the user.

When a frozen PID channel is returned to normal with the UNHOLD command, a "Bumpless Transfer" will occur when the PID calculation resumes. The output will not jump to a new value because the actual PID output had been manipulated by the manual control. Some electronic controllers cannot do this because the manual output is separate from the automatic one. The TEST PID system provided the bumpless transfer automatically when the UNHOLD command returns the PID channel to its normal state.

WARNING: The PID control system contained in the TEST SCADA system relies on digital computer hardware and techniques. Like any other computer based system, the potential exists for hardware and software failures beyond the control of the program. The PID control system and output hardware are not intended for use as a primary safety system. Other means of protecting life and equipment must be provided externally to the computer system in order to prevent injury to personnel or damage to equipment affected by this system. If necessary, external analog backup or bypass systems should be used to provide uninterrupted analog outputs to critical equipment that will be adversely affected by the loss of reliable output from the SCADA system

PID References:

Anderson, Norman A. Instrumentation for Process Measurement and Control*, Chilton Books, 1972.*

Deshpande, Pradeep B., and Ash, Raymond H., Elements of Computer Process Control*, Instrument Society of America, 1981*

Mellichamp, D.A., Real-Time Computing with Applications to Data Acquisition and Control*, Van Nostrand, 1983*

20 FORM DISPLAY FEATURE

SCADAWARE has an easy to use screen form function to simplify the design of custom display screens and user menus. The form operates as a combined display and menu hit processor where realtime data can be presented while menu selections are made with keystrokes or a mouse. This is a completely new feature and is subject to revision at this time. Future enhancements will allow for printing of the forms, but this capability is not present at this time other than with PRINT SCREEN functions.

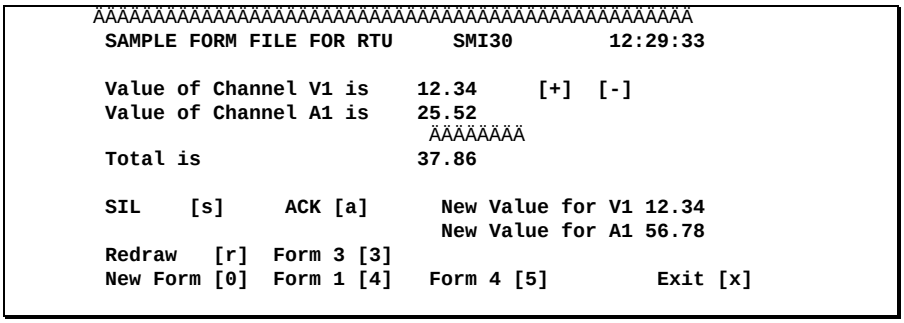
The form is created with the text editor and should have a file type of FRM. The main menu has been modified to allow access to FORM display and editing from the position formally occupied by the USER selection.

The form text file consists of plain text that will form a backdrop for the screen display. Special identifiers are placed on the form to indicate where data values and menu selections will appear. After the form is defined, additional information is provided in the form file to tell the system what is supposed to appear in the data positions, and what is to happen when menu selections are made. Here is a simple form file:

```
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
SAMPLE FORM FILE FIR RTU     [010]            [011]

Value of Channel V1 is    [012]     [+]  [-]
Value of Channel A1 is    [013]
                           ÄÄÄÄÄÄÄÄ
Total is                  [014]


SIL    [s]     ACK [a]      New Value for V1 [1]
                            New Value for A1 [2]
Redraw   [r]  Form 3 [3]
New Form [0]  Form 1 [4]   Form 4 [5]         Exit [x]
?end
?Color blue/gray
010  $R   ; insert RTU name automatically
011  $T   ; update time on display
012  v1`###.##   ; show V1 to 2 decimal places
013  a1`###.##
014  $(V1+a1)`###.##  ; total of 2 channels

a    ACK
s    horn off
X    ? Msg Form Exited Normally
r    ?paint

+    calc v1 = v1 + 1
-    calc v1 = v1 - 1
1    cursor 1,24 : input New V1,v1 : cursor 1,24 : clreol
2    cursor 1,24 : input New A1,A1 : cursor 1,24 : clreol
3    ?GOSUB FORM3   ; gosub to a new form called FORM3.FRM
4     ?RETURN  ; return to previous form
5    ?FORM5   ; branch to form 5 at same nest level
0     ? Form NEWFORM ;quit this form system and start over
;----------------  end of FORM file --------------
```

This small text file does a lot of processing once loaded by the FORM statement. The layout of all the prompts and other text will appear just as it looks in the template above. No X-Y coordinates must be calculated by the programmer. The form designer need only write the text and locate the variable locations with the

bracketed numbers, as in [020]. The bracket notation is simple. Single letters, characters, and digits are place holders for menu hits. Numbers 10 and above are realtime variable locations. The above file would generate a real-time Form display as follows:



The amount of information and menu items available for each screen vary between SCADAWARE and SCADAWARE Lite.

	LITE	SCADAWARE
Screen Value Elements 60		200
Screen Menu Items	30	200
Text Lines	30	100
Nested Screens 4		8

So, each screen can have about menu selections as single letters or numbers (0-9, a-z, +,-,=, etc) and a higher number of real time variables (010-higher).

The space occupied by the real-time values will be determined by their actual size when formatted per system standards. If no format codes are provided, the default decimal places will be used. However, it is best to individually format each value so that no surprises occur when things change later. The form processor will right justify values so that they align on the right bracket. Space to the left is taken up as needed, so leave enough room between the text and data place holder.

Menu hits always take 3 character positions. A menu selection item such as [a] will appear on the screen as <A>, converted to upper case. Selecting the item can be done by pressing "A", or by mouse clicking on any of the 3 character positions in <A>. When the menu selection is made, the form processor immediately executes the statement associated with that menu hit. The form remains on the screen during execution.

The special control characters used within the form file all start with the question mark. Therefore, the question mark cannot be used as a menu hit identifier. The special controls are:

?END	Signals the end of the on-screen portion of the file
?COLOR	identify new screen colors, similar to SET ATTRIB
?PAINT	Menu option to redraw the Form
?EXIT	Menu option to quit the form
?GOSUB	Nested call to another form
?RETURN	Nested return to calling form
?fname	Form name to transfer without nesting

Menu selections with the ? mark are processed differently than ones without the mark. A ? followed by a space tells the form processor that the user wants to exit the form and process the remainder of the menu definition from the normal command prompt. This allows a way to menu-hit out of a form and do something else, including load another form. Use the ?-space combo to leave the form and return to the normal TSP command

prompt.

The ?PAINT statement in a menu hit tells the form processor to redraw the current form. This may be necessary if text has somehow gotten onto the screen, perhaps as a result of an internally processed command.

The ?GOSUB FNAME statement allows another form to be loaded as a subroutine. This is similar to the MENU GOSUB functions in the USER MENU system. if a ?GOSUB is done, the name of the current form is placed in a queue so that it can be reloaded with the subsequent form executes a ?RETURN form statement. This nesting of forms can occur up to 8 deep.

The ?FNAME form statement allows transfer to another form at the current nesting level. A ?RETURN executed in the subsequent form will go back to the original form just as if a transfer had not occurred.

Note that the ESCAPE key can always be used to exit the form system at any time. Any form nesting is lost when the user returns to the command prompt. It is not possible to execute another FORM command within a form itself. To start a new form system, the user must exit the current form and reload a new form with a ? FORM xxxxx menu hit.

FORMs can be started from the main menu with the revised USER/FORM selection. They can also be started from the command line or by any other means with the new TSP command FORM followed by the form file name. For example:

FORM MYSCREEN

will start a form called MYSCREEN.FRM. An file type other than "FRM" can be used, but this is not recommended. For Task 0, the simple TSP command FORM (without a form file name) can be used. This causes a directory menu to be presented of all files with the FRM filetype. An existing form file can be selected with key or menu.

21 MULTI-DROP COMMUNICATIONS

21.1 INTRODUCTION

SCADA systems require the use of some type of communications medium to send data from one unit to another. TEST's SCADA system supports the use of telephones, radios, and direct wire connections to transmit data. The normal process of communication between two units involves a dial out by one unit, a connection with another unit, the transmission of data, and finally a disconnect. This process describes a point-to-point mode of operation where only two devices at a time can communicate. Once a connection is made between two units the communications path becomes blocked and can not be used by other units. Not until the current session is over will other units be able to transmit data.

TEST's RTU/SCADA system also supports a Multi-Drop mode of operation where several units share a single communications path. In this mode any unit can communicate with any other unit as long as the communications path is clear. Unlike the point-to-point mode, the communications path does not remain blocked for an entire session. Instead, the communications path is blocked only when a unit is transmitting. At the end of each spurt of data the path becomes clear. At that time any unit can gain control of the path and transmit its own message to any other unit.

Multi-Drop communications represent a peer-to-peer relationship between units rather than a master/slave relationship. By this we mean that each unit attached to the communications path is equally capable of communicating with any other unit attached to the path. There is no master unit responsible for communication with remote units. Instead, communications can occur at any level, whether it be between a Host and a Host, a Host and an RTU, or simply between two RTUs.

The program supports broadcast mode which allows one unit to send commands to all units on the link. This is suitable for common functions such as outputs off, time set, and daily log-off procedures.

SCADAWARE after 09/93 also supports peer-to-multi-peer links where one unit can send data to more than one unit at the same time. This line TAP mode allows a unit to monitor transmissions intended for other units and process DATA statements only. With the TAP mode, a unit can be a passive listener on a link when operating as a standby master or as a display-only station.

21.2 DCD AND ONLINE STATUS

The TEST RTU/SCADA system uses the serial port of a PC to connect a modem or other data transmission device so that it can send information to a distant unit. The unit on the other end must have a similar device to receive the information and feed it into the computer. For point-to-point communications the Data Carrier Detect (DCD or CD) line is used to determine the online status of the serial communications. Normally, nothing will be transmitted unless the DCD is present. However, for Multi-Drop communications the presence of a DCD will prohibit a unit from transmitting. An explanation of how the DCD is used for all types of communications is given below.

For telephone communications in point-to-point mode a HAYES type modem is used. The DCD is used to reflect the connected state of a modem with another modem. When a modem answers a call from another modem the DCD becomes true at the receiving modem. The calling modem will then detect the carrier from the other end and turn on its DCD. As long as each modem detects an audio tone carrier from the other modem the DCD will be true and the two computers can communicate.

For radio communications in point-to-point mode a special type of modem called a Packet Radio Controller (PRC) is used. The PRC is a little more complicated than a phone modem because of the "half-duplex" nature of radio transmissions. The PRC does not send out a constant carrier tone like the HAYES modem. In the case of the PRC, the DCD status represents a logical connection to another PRC rather than a physical connection. The status of the connection is determined by radio tone handshaking from the modems themselves. When one PRC calls another they exchange codes and the DCD is established. Until the PRCs

disconnect, the DCD will be true even though no actual carrier is being transmitted. The disconnect will occur after a specified period without any activity or when the computer commands the PRC to "hang up".

Direct connect systems for point-to-point communications are fairly simple. Direct connections can be made using a 5 wire null-modem cable. This simply crosses the TD and RD lines from each of the computers and also switches the RTS and DCD lines to provide flow control. The fifth wire is used for the Signal Ground. When a direct connect system "dials" out it simply raises its RTS line and keeps it on for the duration of the session. The other unit will "see" the signal and its DCD input which will cause it to turn on its own RTS line. Therefore, direct connect systems use their RTS lines to activate the DCD inputs of the other computers. As long as the DCDs are present the systems can communicate. Again, a disconnect will occur after a specified period without any activity or when the computer performs a controlled "hang up".

For Multi-Drop communications the DCD line is not used to determine the online status of a unit. The DCD is only used to indicate that another unit is transmitting data and that the communications path is temporarily blocked for use by other units. The online status of a Multi-Drop system is determined by a logical connection rather than a physical one. This is similar to the logical status used by Packet Radio Controllers.

There are two ways that a unit's task can go online when operating in Multi-Drop mode. First of all, a task will logically be placed online whenever an attempt is made to call out. Second, a task will go online whenever it receives a message from the communications line that is intended for that unit. Once online, a task will remain online until a specified amount of time has elapsed without any activity for that unit or a BYE command is processed which forces a "hang up".

In addition to being online, a unit must have a clear communications path in order to transmit. All tasks operating in Multi-Drop mode are continuously monitoring the DCD line of the communications path. If a task wishes to transmit and a DCD is present, the task will have to wait. The amount of time that a task will wait for the DCD to clear is controllable for each task by using the

SET MD DELAY xx xx

command. If the DCD is not cleared before the wait period expires the attempt to transmit will fail and the task will generate the message

COULD NOT SEND

A failure can also occur if the DCD is cleared but another task gains control of the communications path and begins to transmit. The original period that a task will wait for the DCD to clear will not be reset when the DCD first clears. It will continue to count down until it either expires or the task finally gains control and begins to transmit.

When the communications path is clear (no DCD is present) and a task wishes to transmit it will begin by turning on the RTS line. This will cause the modem to turn on a carrier tone that will be detected by the DCD of all other units on the line. A slight delay will occur between the time the RTS line is turned on and the transmission starts. This allows modems and radios to get their carriers turned on before the actual data starts down the line. This also allows the squelch to clear at the receiving radio or modem. This delay is also controllable for each task by using the

SET MD DELAY xx xx

command. (The SET MD DELAY command is explained below.) At the end of the data burst the RTS line is kept on for one additional system tick. This will allow all characters to get transmitted out of the UART without chopping any off.

21.3 RELAXED DCD RULES

Software after 05/93 allows the DCD rules to be relaxed via the SET DCD option. This setting affects

each task individually so that unusual modems and comm paths can be accommodated. With the relaxed mode, the program does not look for a DCD signal at all. It will always be able to send and receive data without checking the DCD signal on the serial port. In some cases, transmission conflicts will occur from time to time. This will be handled by the program with the normal error-check and retry mechanisms built into the software.

Elimination of the DCD requires that a SET DCD OFF statement be placed in the start file for the affected task. This need only be done once, although additional statements in the BYE or other files will not harm the system.

21.4 SOFTWARE SETUP

Setting up a system for Multi-Drop communications is not much different from the setup for point-to-point communications. The main difference is that MULTIDROP must be specified as the communications media. The media to be used by each communications port must be defined when the SCADA program is started. The COMM statement in the main configuration (DAT) file is used to define the type of media used by each port. For example, the line

```
COMM 1 1200 MULTIDROP N 8 2
```

could be used in the main configuration file to setup comm port 1 for Multi-Drop communications with a baud rate of 1200, no parity, 8 data bits, and 2 stop bits.

Normally, the media is setup one time during startup and never needs to be changed. However, in the event that the media does need to be changed once the program is running the SET MEDIA command could be used. The available keywords to be used with this command are PHONE, RADIO, KEYBOARD, MULTIDROP, and NONE. For example, the command

```
SET MEDIA MULTIDROP
```

could be used to change the communications media of a comm port to Multi-Drop at any time.

Another point to consider is that the HAYES and PRC commands required for phone and radio communications in point-to-point mode are not necessary for multi-drop communications. The modems used for Multi-Drop communications are not programmable and do not perform independent processing like the more intelligent HAYES and PRC devices. Multi-Drop modems only serve to convert and detect serial data to and from audio tones. All actions by these modems are controlled by the RTS line of the serial port.

21.5 MODEM MONITOR CHANGES

The Modem Monitor device is a TEST INC. designed and manufactured device intended to assist in unattended operations. This device is used to monitor system activity and perform an automatic reset of the computer if a malfunction occurs in the processor. Although not a communications device itself, it is included here because it ties into the serial port.

The TEST SCADA program uses either the RTS (pin 4) or DTR (pin 20) as a control line for the Modem Monitor device (default is RTS). The software toggles one of these lines once per second to activate the watchdog timer in the Modem Monitor. If the line does not toggle within a specified time period, the Modem Monitor will reset the computer. Refer to the SYSTEM DESIGN manual for more information about the Modem Monitor.

On Multi-Drop systems the RTS line is used as a special signal to control the carrier output of a modem. The software will turn this signal on prior to transmission and will turn it off when the transmission is complete. Because the RTS line is used to control the CD (or DCD) of the modem and the watchdog timer of the Modem Monitor, a conflict results. This conflict can be resolved in one of two ways.

First of all, the RTS can be used to control both functions as long as the computer will transmit often enough to prevent the watchdog timer from timing out. In this case the

MONITOR OFF

command must be used to stop the program from toggling the RTS line once per second. This will allow the Modem Monitor to still use the RTS line without interfering with CD of the modem. Also, the turning on and off of the RTS line during data transmissions will provide the necessary toggling to prevent the modem monitor from resetting the computer.

Second, the control line for the Modem Monitor could be changed to the DTR line. This option should be used on systems where transmissions will not occur often enough to prevent the watchdog from timing out. Changing the control line for the Modem Monitor can be done using the

MONITOR DTR

command. The MONITOR OFF or MONITOR DTR command can be included in the START0.RTU file which automatically gets processed when the program is started.

21.6 MULTI-DROP IDs

In Multi-Drop operation several units share a common communications path. This means that every message transmitted by every unit will be "heard" by all other units. However, each message transmitted is usually intended for only one particular unit. This means that one unit should process the message and all other units should ignore it. Multi-Drop IDs are used to keep track of which units should process which messages.

Each task that is used for Multi-Drop communications has two IDs associated with it. These IDs are included as part of a message header that is attached to each line transmitted by the task. The first ID is used to identify the intended destination of the message. The second ID is used to identify the source of the message. The format of the header is

DestinationID/SourceID/

This header is automatically inserted in the message packet of the sending unit between the leading Control-A character and the command sequence number. For example, consider a task with the Multi-Drop ID GA-HOST that is transmitting a message to another unit whose communications task has the Multi-Drop ID GA343A. The actual message package that is sent might appear as follows:

☺GA343A/GA-HOST/1♠0☺message♥checksum

where 1 is the number of the block being sent and 0 is the number of the block being acknowledged.

By default, each task's Multi-Drop ID is set to the name of the system RTU. This name is defined during startup by using the NAME statement in the main configuration (DAT) file. If this command is not used in the main configuration file the name SYSTEM will be used. Any task can change its Multi-Drop ID at any time by using the **SET MD ID xxx** command where xxx is the name of the new ID. The maximum length of a Multi-Drop ID is 8 characters.

The **DUMP TASK x** command can be used to display a task's Multi-Drop ID. The x is replaced with the number of the task for which to display a list of information.

When a task transmits a message the source ID in the Multi-Drop header is simply taken from the task's Multi-Drop ID. However, the destination ID can come from one of two places, depending on what caused the message to be transmitted. If the message being transmitted is a response to a message from another unit, the destination ID will contain the ID of the unit that sent the original message. On the other hand, if the message being transmitted is the result of an activated link, the destination ID will contain the phone number or

radio call sign of the link. In the case of Multi-Drop communications, the phone number or radio call sign of a link is actually the Multi-Drop ID of the unit to receive the outgoing messages.

21.7 CONFIGURING A MULTI-DROP LINK AND ID

Setting up a link for Multi-Drop communications is very similar to setting up a link for direct point-to-point communications. All link parameters can be programmed using the CONFIG LINK x or the PROG LINK x command. The only difference for setting up a Multi-Drop link is specifying the number to call. Normally, a phone number or radio call sign must be given. This is the actual number that must be called in order for two units to establish a connection and communicate.

However, in Multi-Drop mode there is no actual call out and connect process. Since all units are connected to a common communications path, it is more a matter of identifying the unit to "talk" to rather than calling it up and establishing a connection. For Multi-Drop communications, the phone number or radio call sign that must be specified is actually the Multi-Drop ID of another unit's communications task.

For example, consider a Host and two RTUs that are setup to use Multi-Drop communications. Assume that the Host system is called GA-HOST and the two RTUs are called GA343A and GA343B. Each unit would require a COMM command in its main configuration (DAT) file to set up a comm port for Multi-Drop communications. Each unit would also require an RTU task to be defined which would use the comm port for communications. By default, the Multi-Drop ID for each of these RTU tasks would be the system name.

At the Host two links could be setup to communicate with each RTU. The phone numbers specified for each link would be GA343A and GA343B. At the GA343A RTU, two links could be setup to communicate with the Host and the other RTU. The phone numbers for each link would be GA-HOST and GA343B. The GA343B RTU could also contain two links setup in the same manner. This would allow any unit to be able to activate a link and communicate with any other unit. For example, the first link at the Host could be setup as follows and used to communicate with the GA343A RTU.

```
                                                    LINK   1   Of   4

Link Description                MDrop Link to GA343A
Number of Task Using This Link  1
Phone Number or Radio Call Sign  GA343A
Seconds to Wait For a Connect Before Giving Up                          60
Maximum Number of Tries to Call Out When an Attempt Fails               5
Number of Tries to Rapidly Call Out When an Attempt Fails               2
Seconds to Wait Between Call Out Attempts After Exceeding Rapid Tries   120
Maximum Number of Tries to Call Out Without Resetting Counter to Zero   0
Is This a Host?         Y
Alternate Link  -1




 <F2> Accept • <PgUp> Previous Link • <PgDn> Next Link • <Esc> Cancel
```

NOTE: If a task's Multi-Drop ID is changed using the SET MD ID xxx command, any link set up on another unit which is used to communicate with that task will also have to be changed to the new Multi-Drop ID.

The phone number of a link must specify the Multi-Drop ID of the task it will communicate with, which is not necessarily the name of the unit it will communicate with.

When setting up a link for Multi-Drop communications you must be certain that the phone number (or radio call sign) of the link contains the Multi-Drop ID of the task that you wish to communicate with. Remember, each task has its own Multi-Drop ID that is set to the name of the system RTU during startup. Although this is probably fine, is might sometimes be desirable to change the ID of a task from the default value. This can be done by using the

SET MD ID xxx

command. Again, the common place to use this command is in a task's START file because this command must be processed by the task whose ID you want to change.

If the Multi-Drop ID of a task is changed at one unit all links at other units that are used to "call up" and communicate with that task must be changed. A communications session will be unsuccessful if the activated link's phone number does not match (don't worry about upper or lower case) the Multi-Drop ID of the task at the other end.

21.8 MULTI-DROP SESSIONS

To begin communications in Multi-Drop mode a link must be activated. This can be done by pressing the **F5** key, by using the LINK NOW command, or by using some other link action. When a link is first activated the task that is responsible for processing the link will immediately go online. If a CONNECT file exists for that task it will be processed at that time. The program will then process a LINK file for that link if one exists. If the link file does not exist, the computer will process its own default commands depending on whether the unit that initiated the communications is a Host or an RTU. The default command for a Host unit is BLOCK READ DOWNLOAD and the default command for an RTU is READ DOWNLOAD. Either the LINK file or the DOWNLOAD file will be responsible for processing the commands necessary to transfer the data from one unit to the other.

Most commands that are transmitted between two units require an acknowledgement of some kind. This can be either a response to a command or a simple ACK. For example, a SCAN command requires a response in the form of a DATA command. Other commands simply require an ACK which informs the sending unit that the message was properly received by the receiving unit.

When a message is transmitted that requires an acknowledgement the task that sent the message will remain online and wait for the acknowledgement from the other unit. Although the acknowledgement should be almost instantaneous, the DCD will temporarily be cleared between the time the message is sent and the time the acknowledgement is returned. During that time any unit sharing the communications path is free to transmit.

While waiting for an acknowledgement a task might see messages go by that are transmitted by other units. Some of those messages could contain the Multi-Drop ID of the task waiting for the acknowledgement. This means that other units could be trying to communicate with a task while it is waiting for an acknowledgement from another unit. The task waiting for the acknowledgement will not be able to process any incoming messages other than a message from the unit it is waiting to hear from. Once the acknowledgement is received the task will be able to process other incoming commands that contain its Multi-Drop ID.

What this means is that it is possible for a third unit (or more) to intervene during the communications of two other units. There are only two limitations to processing messages coming from more than one unit at a time. First of all, as already mentioned, no unit can communicate with a task that is in the process of waiting for an acknowledgement or attempting to send an acknowledgement to another unit.

Second, each incoming message must contain a higher block number than previously processed message. If an incoming message contains a block number less than the last processed message, a simple ACK will be returned because the task will assume that it has already received and processed that message.

The reason for this is that each task only keeps a single sequential block counter for all incoming messages. The source of the messages does not matter because a separate block counter is not kept for each unit sharing the communications path.

An additional point to take note of regarding incoming block numbers is the special case of block number 1. Except for the first incoming message of a session, a task will reset its incoming and outgoing block counters to zero when it receives an incoming message with a block number of 1. Normally, incoming messages start with a block number of 1 and are incremented by 1 for each additional incoming message. When a message comes in with a block number less than the current block count the message is ignored and an ACK is returned. However, if the incoming block number is 1, the message will get processed and the block counters will get reset. This feature allows a task to automatically reset the block counters of another unit without having to process a BLOCK RESYNC command. This is common for situations where a task processes a file that continuously loops sending constant updates to another unit. At the end of the loop the file could contain a RESYNC command to reset the block counters locally. When the next message is sent to the other unit it will contain the block number 1. When the other unit receives a message with the block number 1 it will reset its own block counters.

Block counters are also reset for a task each time the task performs a disconnect. Upon going offline a task's block counters are reset to 0. Each time the task goes online again it will start its outgoing block count at 1.

21.9 BROADCAST MODE

Communications in Multi-Drop mode begin when a link is activated and one unit attempts to transmit data to and receive data from another unit. The unit to be contacted is determined by the Multi-Drop ID that is specified for the phone number or radio call sign of the link that was activated.

During normal Multi-Drop communications a message is sent out by one unit. That message is seen by all units sharing the communications path. However, only one task monitoring the path should have the same Multi-Drop ID that is contained in the message. That task is responsible for processing the message and returning some type of response to the sending unit.

The RTU/SCADA program also supports another mode of operation for Multi-Drop communications. This mode, known as BROADCAST MODE, occurs when the Multi-Drop ID (phone number or radio call sign) of an activated link is 0. By specifying 0 as the Multi-Drop ID for a link, instead of an actual task's Multi-Drop ID, a single unit is able to communicate to all other units on the path at the same time.

Normally, each unit monitoring the communications path will only process a message that contains its Multi-Drop ID in the message header. However, each task will also process any incoming message that has a Multi-Drop header of 0 as the destination ID. This allows a single command to be sent out by one unit that will get processed by all other units.

Although this seems simple enough, there is one catch to broadcast mode. When a task receives a message in broadcast mode it will process the message but never return any type of reply to the sending unit. This means that the sending unit is never sure of what units received the message properly and processed it and which ones did not.

This may seem like an unreliable mode of communication at first, but when used properly it poses no real problem. Broadcast mode is designed to be used in situations where one unit sends the same group of commands to all other units over and over again in a continuous loop. Therefore, if a message does not properly get transmitted to some units it shouldn't matter. It should only be a matter of seconds before the same command will be transmitted again.

21.10 DATA TAP MODE

Software after 09/93 has the ability to "tap" onto a multi-drop comm line and process DATA statements addressed to other units. This is very helpful in systems that have more than one computer acting as a Host or display station. With this feature, a computer can listen to data transmissions and receive updates from a number of RTU's without ever having to be directly addressed by them. This allows the Tapping computer to be constantly updated for display purposes, and also simplifies multi-master systems where a standby unit must be kept up to date with data from a number of RTUs.

The TAP configuration requires that memory be allocated for the list of acceptable System names that will be monitored. This is done in the system's DAT file with a simple TAP nn statement that tells the system we will be monitoring a maximum of nn system address Ids. Note that RTU names are not what is monitored. TAP mode looks for system address names in the first part of the Multi-Drop header. If the message is intended for an ID that is on the acceptable list, then the Tapping computer processes the statement as if it were Broadcast to all units.

Tapping is very similar to Broadcast in that no reply is generated by the receiving unit. The difference is that Broadcast mode allows any command to be processed. Tap mode only processes DATA statements. Therefore, commands from one unit to another that control procedure processing, calculations, and other TSP functions are ignored by the Tapping unit.

The list of System Ids to be Tapped must be entered once during startup, normally as part of the START0 operation. The TAP settings are global to the entire computer (i.e. all tasks) and do not need to be done separately for each task. The normal sequence is as follows:

```
TAP Clear  ; elim any old taps
TAP ADD WC450 HOST VR247 EC311 MP290
```

The TAP Clear command initializes the Tap list to empty. The TAP ADD line provides the System names to be monitored. More than one TAP ADD line can be used to list the acceptable names. An additional command, TAP DUMP, provides a short display showing the names currently on the list.

Tasks that are "Watched" by another task will show all comm lines entering the serial port and will provide an additional TAP= line to show that the line was processed as a data tap. This allows monitoring of the TAP operation to insure that the system is processing the desired commands.

21.11 MULTI-DROP TIMING SETTINGS

Multi-Drop communications have timing requirements that must be specified to provide optimum performance in each individual system. SCADAWARE provides for two timing parameters: The key-up delay, and the communications-busy timeout. The SET MD command is used to specify both of these timing parameters.

The syntax is:

`SET MD DELAY  dcd_wait_seconds  keyup_delay_ticks`

The default for dcd_wait_seconds is 5 seconds. Keyup_delay_ticks defaults to 9 ticks, approximately 1/2 second. Although both these defaults will work in most systems, it may be necessary to change them for particular needs.

For example, it is possible for the communications path to be busy because several units are trying to transmit simultaneously. In complex systems, it is necessary to increase the DCD wait period. This will give a task a little more time to wait for the DCD to clear before giving up. It may also be necessary to increase the delay between keying the communications device (i.e. modem via RTS) and the start of transmission.

The following example sets the DCD wait to 10 seconds, and the key_up delay to 1 second:

SET MD DELAY 10 18

The DCD wait period and the keyup delay period are independently set for each Multi-Drop task. This means that the SET MD DELAY command will affect only the task that processes the command. A convenient place to use this command (if it is necessary at all) is in a task's START file (START1.RTU, START2.RTU, etc).

The SET MD DELAY command can be used without any parameters to display the current settings for a task. Again, the task that processes the command will be the task that the settings are displayed for. The "watch" will have to be turned on to actually see the current settings displayed for any task other than task 0 (the local CRT). For example, the following sequence of commands could be used to display the current Multi-Drop delays for task 1.

```
WATCH 1
FORCE 1 SET MD DELAY
WATCH OFF
```

21.12 TERMINAL MODE

In normal point-to-point communications the program can operate in two modes, RTU mode and Terminal (Human) mode. In terminal mode simple text messages are transmitted from one computer to another. For phone and direct connect communications characters are sent one at a time from one computer to the other. When the receiving computer receives a carriage return it knows that the line is complete and ready for processing. For radio communications the characters are buffered in the Packet Radio Controller and sent as a single block when a carriage return is entered. The PRC is responsible for keying up the radio and transmitting the message.

In RTU mode the same type of messages are transmitted as in Terminal mode but each message is automatically packaged with special codes and sent as a single block. These special codes are used to control message sequencing and detect errors in data transmissions. When the receiving computer receives a message it will perform all the proper error checking before processing the command. This is the normal mode of operation for all systems.

When using Multi-Drop communications only the RTU mode of operation is available. In RTU mode the program will automatically package a message with all of the special error checking codes, wait for the DCD to clear, turn on the RTS, pause for the keyup delay, transmit the message, wait an additional system tick, and finally clear the RTS. In this case the program is responsible for keying up the modem prior to transmitting. The modem does not take care of keying itself up as it does for radio communications in point-to-point mode.

The Terminal mode of operation will not work for Multi-Drop communications because the program does not key up the communications device each time a character is entered nor does it buffer all characters until a carriage return is entered. When attached to a task that is used for Multi-Drop communications the *TERMINAL command can be used to put the task into Terminal mode. As each character is typed it will immediately be sent to the communications device to be transmitted. Since there is no buffering and no key up the characters will simply be lost rather than transmitted.

21.13 CHECKSUM ERRORS

As already mentioned, each message that is transmitted is automatically packaged with special codes before it is sent. These codes are used to check the sequence of incoming and outgoing messages as well as detecting any problems with the actual transmission. To check the validity of a data transmission a special checksum code is attached to the end of every outgoing message. This checksum is calculated from the actual character string that is being transmitted.

When a task receives an incoming message it will calculate its own checksum based on the character string it received. If the character string is scrambled during the transmission and what is received is not actually

what was sent, the checksum will catch the error.

Any time a task receives a message and detects a checksum error the message will not get processed. In normal point-to-point mode communications the receiving task will send a NAK to the sending task to inform it of the error. The sending task will immediately attempt to send the message again.

For Multi-Drop communications the process is a little different. First of all, if the message gets scrambled during transmission it might contain an invalid Multi-Drop ID when it is seen by the tasks monitoring the communications path. In this case no task will process the message, the sending task will time out waiting for an acknowledgement, and finally the sending task will attempt to resend the message.

Another possibility is that the message gets scrambled during transmission but the Multi-Drop ID of the message still matches the ID of one of the monitoring tasks. Again, no task will process the message and no NAK will be sent back to the sending task. This will also cause the sending task to time out waiting for an acknowledgement and finally attempt to resend the message. The reason the task does not return a NAK (as is done in point-to-point mode to force an immediate resend) is because there is no way to be sure that the message was originally intended for the task whose Multi-Drop ID the message contained.

21.14 CALL-ON-EXCEPTION ALERT COMMAND

TEST's smaller RTUs have call-on-exception capability initiated with the ALERT command. This command is normally broadcast from a small RTU onto a multi-drop link for processing by one Host on the network. The form of the ALERT command is ALERT TAGNAME, example:

```
ALERT   RTU2
```

This command, sent from the small RTU, will be processed as a broadcast message by all other units on the link. If ALERT is enabled on the receiving computer, the task will scan the LINK table to try and match the TAGNAME passed with the ALERT command. If the tagname is found, the corresponding LINK will be activated. This allows small RTU's to announce that they need attention without specifying the nature of the problem at the remote location. Activation of the proper link at the Host will cause the Host to poll the RTU, thereby determining the reason for the call.

The small RTU will continue to broadcast ALERT commands every minute for one hour. The Alert status is canceled at the small RTU when it receives a poll request from a host.

Note that only one Host on the network can respond to the Alert request. Multiple responses would clash on the multi-drop network and cannot be allowed. Any Host tasks that are required to process Alert requests must execute SET ALERT ON sometime during their startup. Note that this setting is unique to each task, so it is possible to have one multi-drop task be the Alert processor while other multi-drop tasks on the same computer do not.

22 AUDIO MESSAGE PLAYBACK

22.1 SOUND PLAYBACK INTRODUCTION

SCADAWARE has the ability to play back audio messages using standard PC based sound I/O cards. These cards include Soundblaster, Microsoft Sound System, and many other similar devices. The recording of the messages is done using utility software provided with the sound card. Playback is handled directly by SCADAWARE using drivers loaded before the SCADAWARE startup process.

Playback of messages can be done manually, through menu selections, or automatically when a channel goes into the abnormal state.

NOTE: This section of the manual is not complete. Additional information on the sound playback system will be provided in the next release of the manual, or as an addendum to this manual.

22.2 AUDIO SOUND FILES

Audio sounds are stored in separate DOS files, normally in Microsoft WAV (wave) format. Recording of sounds is not done in SCADAWARE. Digitized sound images are generated by software provided with the sound card.

Each sound phrase is stored in a separate DOS sound image file that is given a unique name. Multiple phrase files can be announced in sequence to form complex messages. The sequential announcement can be specified directly by listing the file names one after another in a single PLAY command. Files can contain individual words or complete messages. Typical sound files may be as follows:

FILENAME	"Word"
LOW.WAV	Low
FLOW.WAV	Flow
RATE.WAV	Rate
COMPRESS.WAV	Compressor
SHUTDOWN.WAV	Shut Down
METER.WAV	Meter
1	One

Each of these files represents a single word. They can be combined into meaningful messages by combining them in sequence. Care must be taken during recording to establish a common tone and cadence to the words so that they will sound smooth when played back in random order. This gives a somewhat "robotic" sound to the messages which may be undesirable in some cases. To avoid the monotonous tone, record complete multi-word messages in single files and play them back as a single entity.

Note: WAV files must start with a letter, not a number. This is because the PLAY command (described below) looks for numeric values and treats them differently than normal file name references. Each system should have WAV files for the digits 0-9, plus one for the word "POINT." These are used in numeric announcements.

The audio WAV files are normally stored in a separate directory which must be specified with the SET PATH SOUND XXXXXX command. Once specified, the PLAY command will use this directory as the default source for all sound files. To override this specification, enter the complete file name for the desired file, as in:

```
PLAY C:\AUDIO\HELLO
```

22.3 PLAY COMMAND

The PLAY command is used for playback of the prerecorded sounds. Sound files can be played individually, or in "strings" of phrases to form complex messages. The format of the sound files is completely up to the user, and can therefore be in any language or combination of languages.

PLAY will seek out and playback sound messages according to the file names provided on the command line. It can also be directed to use a command line parameter as a "lookup" into a list which specifies a list of phrases. This lookup function allows for a single word, such as a channel tagname, to represent a predefined sequence of sound files.

PLAY can be executed in the Local task or in a background task. PLAY will take all of a task's resources during the entire playback, so it is best to use the Util task instead of Local. However, the Local task can send messages (through menu hit or direct keyboard entry) to the Util task, allowing the Local user to start background playback.

To syntax to play a single sound file is PLAY sound_file_name. To play back multiple messages, simply enter the additional names on the command line as in:

```
PLAY COMPRES NUMER 1 SHUTDOWN  ; play 4 message files.
```

To have the system lookup a tagname, the syntax is:

```
PLAY /TAG tagname  ; lookup phrases for tagname.
```

The PLAY command will automatically decode numeric values into a sequence of phrases, one for each digit. For example, the command line PLAY 123 causes playback of 1.WAV, 2.WAV, and 3.WAV. SCADAWARE expects WAV files to exist for each digit, 0-9, plus one named POINT.WAV which contains the word Point (used for decimal places). This is useful for phrase elements that are generated at runtime.

Example: Assuming channel A5's value is 43.7, the command line

PLAY WELL PRESS IS $(A5) PSI

will be expanded at runtime to look like:

PLAY WELL PRESS IS 43.7 PSI

The $(A5) was expanded into the proper numeric format and passed to the PLAY command for announcement. PLAY will pick apart the number and say each digit individually, such that the announcement may sound like "Well Pressure Is 43 Point 7 PSI."

See the PLAY command description in the TSP COMMAND REFERENCE for more information on options to the PLAY function.

22.4 TAGNAME PLAYLIST FILE

Sequential phrase announcement can be done indirectly by associating a list of phrases with each data channel. With the indirect method, a single channel tagname is used as a lookup into a text file list which specifies the sound phrases for each data point. The single tagname is used in place of the complete list of phrases desired.

The text file name used for indirect announcements must have name of the RTU and a file type of .PLY. Thus, the Play List file name for an RTU named WC240 would be WC240.PLY. That file contains a simple line-by-line list of the channel tag name followed by the phrases assigned to that point.

A short file for an RTU named GASMET1 may look like this:

```
; FILE GASMET1.PLY
; Play List for RTU at Gas Meter One (GASMET1)
DIFF1 Low Flow Rate
COMPSD      Compress Shutdown
METER1      Low Meter 1 Rate $(M1)
```

This file defines message strings for three separate RTU data points: DIFF1, COMPSD, and METER1. Note that several WAV files are used more than once, demonstrating how sound files can be combined and used for more than one complete message. Note also that a runtime expression $(M1) is used for the METER1 message. The $(m1) will be expanded at the time of announcement into a decimal number, such as 16.5, and will be automatically announced as "ONE SIX POINT FIVE." Any numeric value in the PLAY string will be said as a series of numbers. This requires that the numbers 0-9, and the word POINT exist in the library containing the prerecorded messages.

When a point set to PLAY on Abnormal first goes into the abnormal state, a message is sent to the Utility task in the form of:

PLAY /TAG RTUNAME.TAGNAME

This causes the current utility task to start the PLAY command in /TAG mode where it expects to find a file named RTUNAME.PLY with an entry for point TAGNAME. If the file exists, and the tag name is matched, then PLAY will work with the sequence of phrase messages specified in the file. If no matches are made, then the command is ignored.

It is useful to monitor the Util task (WATCH UTIL) during playback to see errors and status reports for the playback system.

22.5 PLAY PUSH-TO-TALK

Any channel can be assigned as the push-to-talk (PTT) channel, although this will almost always be a status output connected to a Public Address (PA) system or radio system. If a PTT is assigned, SCADAWARE will calc the channel to 1, wait one second, say the phrases, and then calc the channel back to 0. This will accommodate sound amplifier and radio systems that require a control signal to turn them on and off.

See SET PTT in the TSP COMMAND REFERENCE for information on how to set up the push-to-talk channel.

23 FAX REPORT OPTION

23.1 INTRODUCTION

The full version of SCADAWARE can generate and transmit fax reports over phone lines using a computer with a minimum 386-33MHz processor, 4 Mb of RAM, and a Hayes compatible PC fax modem (Class 1 or 2).

23.2 FAX GENERATION

All fax reports are dot-by-dot representations of text characters contained in a SCADAWARE report file. The The following steps must be performed to generate and send a fax report:

1. Create a standard ASCII text file of the report to send.
2. Create a fax image file from the standard ASCII text file using the FAX MAKE command.
3. Send the fax image file using the FAX SEND command.

Standard text file reports can be created a number of ways. The most common are to use the REPORT command to create generic data reports or to use the FILE command followed by several WRITELN commands to create custom data reports. An example of each is illustrated below in the sample TSP procedure.

23.3 FAX COMMAND

The FAX command is the basis for all fax operations. Available keywords and additional parameters that can be used with the FAX command are shown below:

FAX MAKE fname — Create a fax image file from a standard ASCII text file specified by FName. No specific extension is expected for the ASCII text file name. The resulting fax image file will consist of the same name as the text file and the extension ".APF".

NOTE: During the conversion from text to fax format the program uses a default font style that is contained in a separate file titled APFAX.FNT. This file must be present in the default SCADAWARE directory in order for the conversion to be successful.

FAX SEND fname — Make a call and send the fax image file specified by FName. The extension for the given file name is assumed to be ".APF" and will be used regardless of whether or not the given name has an extension.

FAX DUMP — Display the current fax setup options.

FAX OPTION /[I|T|M|H|S|N|B|F|L]
Provide setup information used by the fax processor during transmission. These options can be specified during startup or at any other time before the FAX SEND command is processed. Each option is designated by a slash (/) or a dash (-) followed by a letter and possibly additional information. More than one option can be specified on a single line, although this is not recommended for clarity reasons. An explanation of each option is given below:

/I xxx — Transmitting fax ID which appears at the top of each page. The default ID is "DEMO HOST/RTU".

/T xxx — Title which appears at the top of each page. The default title is "SCADAWARE FAX REPORT".

/M xxx — Modem inititialization string which is unique to each fax modem. The default string is "V0Q0

&K6".

/H Enable high-level fax functions if available.

/S 1|2|A|C Set fax class to 1, 2, Auto or CAS. The default class is 2.

/N xxx Phone number to call when FAX SEND is processed. The default phone number is blank.

NOTE: This option is not required when using a SCADAWARE communications link to send a fax. This is because the fax phone number is automatically obtained from the phone number specified in the link setup. The link phone number can be overriden in the LINKx file with the FAX OPTION /N command.
This command, if used, should be placed in the LINKx.RTU file just before the FAX SEND command.

/B nnn Modem baud rate (not fax data rate). The default baud rate is 19200.

/F nnn Fax data rate. The default fax data rate is 9600.

/L On|Off|Fname
On or Off will enable or disable automatic logging of each attempt to send a fax and the result status of each attempt. The default state is Off. Fname is the name of the file where the fax logs will be stored. The default log file name is "FAX.LOG".

A few examples of the FAX OPTION command are as follows:

```
FAX OPTION /F 14400          ; change fax data rate
FAX OPTION, /N, 1-800-555-1212   ; set phone #, note commas for delims
FAX OPTION /S 1              ; set fax modem class to 1
```

23.4 FAX REPORT GENERATION

The first step towards generating and transmitting a fax report is to create a file which contains all the necessary commands for creating a report in the standard text file format, converting the text file into a fax image file, and sending the fax. The following is an example of a file which illustrates how to produce and send a fax report.

```
; Demo to make and send a fax report
set eject off               ; no form feed at the end of a report
set delim space             ; put spaces between writeln outputs

sele mo914                  ; make MO914 current RTU report to faxtest.txt
     ; create new text file of report for MO914

sele mo870                  ; make MO870 current RTU
report append faxtest.txt   ; add report for MO870 to existing text file

file append faxtest.txt     ; open text file for additional information
writeln
writeln
writeln, End of Fax Report Generated at $T on $D
writeln
writeln, SCADAWARE by TEST Inc.
file close                  ; close text file with all report data

fax make faxtest.txt        ; create fax image file FAXTEST.APF

fax option, /I, Main Field Computer
fax option, /N, 9^371-3160  ; phone #, note use of ^ in place of comma
```

```
fax send faxtest.apf        ; make call and send the fax image file

bye
```

After this TSP procedure file is created, it can be used to generate and transmit a fax report with a SCADAWARE telephone communications task. For example, assume that the above TSP commands are saved in a file named SENDFAX.RTU and that task 1 is setup as a telephone communications task. At any time the command

FORCE 1 READ SENDFAX

could be used to have task 1 generate and send the fax report. Although this will work, it is not the best way to send a fax because it provides no error checking or means for trying again if the transmission is unsuccessful.

23.5 FAX LINKS

The preferred method of processing fax transmissions is to designate a normal SCADSWARE link as a FAX link. This uses the link to initiate generation and transmission of a fax report whenever the link is activated. To setup a link as a fax type link simply answer YES to the question, "Is This a Fax Link?", which is found on the link configuration screen. Notice that the answer to the question, "Does This Unit Act as a Host When Calling Out?", has no affect for a fax type link.

When a fax type link is activated, it will immediately look for and begin processing the corresponding LINKx.RTU file. At no time will the link try to perform a call out on its own. The actual process of generating a fax report, calling out and transmitting the fax is controlled by the LINKx.RTU file.

If the proper LINKx.RTU file does not exist or a an attempt to fax is unsuccessful, the link will automatically time down for a specific amount of time and then try again. It will repeatedly do this until the fax is succussfully transmitted or the maximum number of tries is exceeded. The timing between each attempt will begin immediately if the proper LINKx.RTU file does not exist. Otherwise, the timing begins when processing of the LINKx.RTU file is completed.

The amount of time a link will wait between each attempt to fax is determined by two parameters found on the link configuration screen. The callout delay is only 30 seconds until the "Number of Tries to Rapidly Call Out When an Attempt Fails" is exceeded. Subsequent delays are determined by the "Seconds to Wait Between Call Out Attempts After Exceeding Rapid Tries." If the rapid 30 second delay between attempts is not long enough, then the "Number of Tries to Rapidly Call Out" should be set to 0.

The following is an example LINKx.RTU file that could be used to generate and send a fax report:

```
; Demo to make and send a fax report
if @curtry(-1) = 1          ; first callout attempt since link activated
  set eject off             ; no form feed at the end of a report
  set delim space           ; put spaces between writeln outputs

  sele mo914                ; make MO914 current RTU
  report to faxtest.txt     ; create new text file of report for MO914

  sele mo870                ; make MO870 current RTU
  report append faxtest.txt ; add report for MO870 to existing text file

  file append faxtest.txt   ; open text file for additional information
  writeln
  writeln
  writeln, End of Fax Report Generated at $T on $D
  writeln
  writeln, SCADAWARE by TEST Inc.
```

```
  file close                ; close text file with all report data

  fax make faxtest.txt      ; create fax image file FAXTEST.APF
endif

fax option, /I, Main Field Computer     ; ID printed on top of each page

fax send faxtest.apf        ; make call and send the fax image file

if @faxerr(0) = 0           ; if no error sending the fax
  link reset                ; clear link so it won't try again
endif
bye
```

23.6 @CURTRY and @FAXERR

Notice that there are only minor differences between the LINKx.RTU file and the SENDFAX.RTU file shown above. The LINKx.RTU file uses the @CURTRY(-1) function to determine how many times the attempt to fax has been made since the link was activated. Only the first attempt should have to create the report and convert it to a fax image file. All other attempts can skip this and proceed directly with transmission of the fax.

NOTE: The @CURTRY(x) function uses parameter -1 to return the current number of tries for the current RTU's current link. A parameter value between 1 and the maximum number of links can be specified instead to check the current number of tries for a specific link.

Fax reports generated with a LINK file do not require specification of the phone number with the FAX OPTION /N command. The phone number is automatically obtained from the phone number specified on the link configuration screen. However, the FAX OPTION /N command can be used for special cases to override the phone number nornally supplied by the associated link.

The last difference in the LINKx.RTU file is that the @FAXERR(0) function is used check the status of each fax transmission and reset the link when a successful transmission has been completed. If this check were not included the link would repeatedly time out and resend the fax until the maximum number of tries is exceeded.

The @FAXERR(0) function should be called immediately after each FAX SEND command. If the communications port is invalid, the fax number is busy, or there is any other problem that prevents the fax from being transmitted, the @FAXERR(0) function will return a unique non-zero error code. Only when a fax is successfully transmitted will this function return a 0. Possible error codes returned by the @FAXERR(0) function are as follows:

2923 - Timed out waiting for data
2926 - Fax aborted by user
9800 - Voice answered phone
9801 - Data modem answered phone
9802 - Line was busy
9803 - Font file not found
9805 - Error initializing modem
9806 - Error during modem training
9807 - Error during session
9999 - Tried to send a fax to comm port 0
9998 - Tried to send a fax to an invalid comm port
9997 - Error initializing fax object
9996 - Failure to identify modem
9995 - Failure to set fax class

The @FAXERR(0) function returns the value of a global variable that contains the status code of the last

FAX SEND command processed by the system. There is not a separate fax error variable for each task. Likewise, all fax parameters that can be set using the FAX OPTION command are also global. Therefore, you must use extreme caution when using more than one task to send faxes at the same time. If two tasks attempt to send a fax at the same time, the parameters set by one task can be overwritten by the other task before any fax is actually transmitted. Also, due to task switching between the FAX SEND command and a call to the @FAXERR(0) function, the @FAXERR(0) function processed by one task can actually return the result status of the fax attempt made by the other task.

Whenever a FAX SEND command is processed it tries to use the communications port associated with the task that is processing the command. Although each port has a set baud rate that is used for normal communications, that rate is temporarily changed during the start of FAX SEND command and is automatically returned to the original rate when the command is completed. The default modem baud rate that is used during fax transmissions is 19200. This rate can be changed using the FAX OPTION /B command which can be processed during startup or at any other time before the FAX SEND command is processed.

23.7 SAMPLE FAX REPORT

The following is an example fax report that could be generated by the SENDFAX.RTU and LINKx.RTU files shown above:

```
01/16/96  04:02pm   SCADAWARE FAX REPORT  From: Main Field Computer Page 1 of 2

MO914 - SANTA FE MO914 RTU                      AT 13:02:29  01/16/96  Page 1

-------------------------- STATUS INPUT CHANNELS ----------------------------
CH  TAG      CHANNEL NAME         LATCHED-STATUS    ALM TIME  ALM DATE    ALM
1   S1       ESD/Fusible Loop     In Service        00:00:00  01/01/00
2   S2       Well #1 Status       Open              00:00:00  01/01/00
3   S3       Well #2 Status       Open              00:00:00  01/01/00
4   S4       Well #3 Status       Open              00:00:00  01/01/00
5   S5       Generator Status     Running           00:00:00  01/01/00
6   S6       Gas Detector 20% LEL Off               00:00:00  01/01/00
7   S7       Gas Detector 60% LEL Off               00:00:00  01/01/00
8   S8       Fire/Smoke Detector  Off               00:00:00  01/01/00
-------------------------- ANALOG INPUT CHANNELS ----------------------------
CH  TAG      ANALOG INPUT NAME        V A L U E            CTRL           ALM
1   A1       RTU Battery                12.9 Volts
2   A2       Nav-Aid Battery             0.1 Volts                        New
3   A3       Fog Horn Battery            0.0 Volts                        New
4   FTP1     Well-1 FTP                    0 Psi
5   FTP2     Well-2 FTP                    0 Psi
6   FTP3     Well-3 FTP                    0 Psi
7   STAT1    Gas Sales Static              0 Psi
8   TEMP1    Gas Sales Temp              0.0 Deg-F
9   DIFF1    Gas Sales Diff              0.0 Inches
-------------------------- AGA-3 METER CHANNELS ----------------------------
CH  TAG      AGA-3 METER NAME           RATE           PLATE   CTL   ERR  ALM
1   M1       Gas Sales                  0.00 MMCF/Day  3.000           0
------------------------- TOTALIZER CHANNELS ----------------------------
CH  TAG      TOTALIZER NAME          TOTAL             CURR  --STARTED--  ALM
1   TGAS     Daily Sales             0.000 MMCF       0.000  13:43  01/18

                Report generated at 16:00:24  on 01/16/96
                          SCADAWARE by TEST Inc.
```

```
01/16/96  04:00pm   SCADAWARE FAX REPORT  From: Main Field Computer Page 2 of 2

MO870 - SANTA FE MO870 RTU                      AT 13:05:32  01/16/96  Page 1

-------------------------- STATUS INPUT CHANNELS ----------------------------
CH  TAG      CHANNEL NAME         LATCHED-STATUS    ALM TIME  ALM DATE    ALM
1   S1       ESD/Fusible Loop     In Service        00:00:00  01/01/00
2   S2       Well Status          Open              00:00:00  01/01/00
3   S3       Generator Status     Running           00:00:00  01/01/00
4   S4       Gas Detector 20% LEL Off               00:00:00  01/01/00
5   S5       Gas Detector 60% LEL Off               00:00:00  01/01/00
6   S6       Fire/Smoke Detector  Off               00:00:00  01/01/00
-------------------------- ANALOG INPUT CHANNELS ----------------------------
CH  TAG      ANALOG INPUT NAME        V A L U E            CTRL           ALM
1   A1       RTU Battery                12.9 Volts
2   A2       Nav-Aid Battery             0.0 Volts                        New
3   A3       Fog Horn Battery            0.0 Volts                        New
4   FTP      Well FTP                      0 Psi
5   IFP      Incoming Flow Press           0 Psi
6   STAT1    Gas Sales Static              0 Psi
7   TEMP1    Gas Sales Temp              0.0 Deg-F
8   DIFF1    Gas Sales Diff              0.0 Inches
-------------------------- AGA-3 METER CHANNELS ----------------------------
CH  TAG      AGA-3 METER NAME           RATE           PLATE   CTL   ERR  ALM
1   M1       Gas Sales                  0.00 MCF/Hr    3.000           0
------------------------- TOTALIZER CHANNELS ----------------------------
CH  TAG      TOTALIZER NAME          TOTAL             CURR  --STARTED--  ALM
1   TGAS     Daily Sales             0.000 MMCF       0.000  13:43  01/18

                Report generated at 16:01:02  on 01/16/96
                          SCADAWARE by TEST Inc.
```

```
End of Fax Report Generated at 16:01:04 on 01/16/96
SCADAWARE by TEST Inc.
```

-end-