TOTAL ENGINEERING SERVICES TEAM INC.

RTU/SCADA SOFTWARE

SYSTEM SETUP AND PROGRAMMING GUIDE

Aug 19, 1991

Document Number 1030-01

P.O. Drawer 1760
671 Whitney Ave.
Gretna, La. 70054
(504) 368-6792

TOTAL ENGINEERING SERVICES TEAM INC.

RTU/SCADA SOFTWARE

SYSTEM SETUP AND PROGRAMMING GUIDE

INTRODUCTION

This document presents programming guidelines for TEST's SCADA systems. The purpose is to provide a common method of setting up and executing typical procedures so that all systems will have similar operational patterns. Although many of the procedures presented here could be done many ways, compliance with these guidelines will result in an easier to maintain system because the system will be similar to most other units.

Users supporting their own systems do not have to comply with these guidelines if they choose to develop their own methods. However, all systems supported by TEST will be designed with these concepts in mind. Existing systems will be modified as needed to bring them in line with these policies.

This document assumes some knowledge of SCADA systems in general, and TEST's system in particular. Detailed information on the commands and procedures mentioned can be found in the SYSTEM DESIGN CONCEPTS and COMMAND REFERENCE documents.

SYSTEM DESIGN

The TEST SCADA system is a Personal Computer (PC) based telemetry system that can be used for a variety of remote data gathering and control purposes. There are several levels of setup and programming required to tailor the system for each location. The many options available often interact with one another, so a complete understanding of the system requirements must be defined so that these options can be set correctly. Some options relate to system configuration, others relate to I/O point definition, and others relate to procedure programming. Each of these topics will be covered in this document to illustrate standard methods of building a system.

STYLE, COMMENTS, AND UPPER/LOWER CASE

Although a subject like "upper and lower case " is not the most important thing about programming the SCADA system, it is being mentioned up front because it reflects the overall tone of this discussion on programming style. After all, the program almost never cares about upper and lower case as either will work in most instances. Well, there is a lot more to learn about setting up a SCADA system than simple typing style.

There are many ways to set up a system, many of which will do the job, Our goal here it to do more than just the minimum job, but rather to provide an

easily maintained system that is consistent with many other systems. The reason for this is more than mere esthetics. One clear mark of a professional programmer is that the programs he creates are neat, clearly understood, and easily followed. A "hacker" will whip up some quickie that will do the job, but it will not be understood by anyone, including the original programmer, in just a short time.

Consider these examples taken from actual installations that show various methods of performing a remote unit shut-down:

BAD EXAMPLE

```
PROC SX
sele ec320
dial
wait 30 conn
set online on
block calc t1 = 30
bye
```

BETTER EXAMPLE

```
PROC shut_in ; Proc to call EC320 and pulse shutin output
  sele ec320
  dial
  msg Waiting for connect
  wait conn 30
  set online on
  msg RTU is online.  Sending Pulse command
  block calc t4 = 30
  sleep 30
  block read download
```

BEST EXAMPLE

```
PROC Pulse_Timer  ; Proc to pulse any output timer.  Param 1
                        ; is the RTU, Param 2 is timer, 3 is secs.
  sele $1
  msg Calling RTU $R to pulse $2 for $3 Seconds.
  dial
  wait 30 conn
  if @comstat(1)
 msg RTU is online.  Sending Pulse Command.
else
 msg RTU did not connect.  Retry later.
endif
   set online on
block calc $2= $3
   Msg Sleeping for $3 seconds to allow timer to complete
   sleep $3   ; Give RTU time to react
   Msg Getting an RTU download
   block download  ; Transfer control to the RTU
```

The first example will work, but has some limitations and uses some poor practices. First of all, there are no comments associated with the procedure. The

procedure name, SX, means nothing to everyone except the original author. The destination RTU, timer, and number of seconds are all "hard coded" into the program, and can only be used for that one purpose. Also, there are no status messages sent to the console that tell the operator the status of the procedure.

The second example is a lot better because it uses a meaningful procedure name, "shut_in". This will tell any follow-up programmer that this proc is about shutting something in. The proc has a few comments to explain its operation, and sends a few status messages to the console. However, this example is still restricted to a single location and timer channel.

The last example is a reusable "generic" procedure that can be used to call any RTU, and pulse any timer for any number of seconds. This is done with the use of command line parameters that will appear on the line when the procedure is called. An example would be:

```
GOSUB Pulse_Timer VR453 T4 30
```

The three parameters are "VR453", "T4", and "30", and they correspond to $1, $2, and $3. The program will pass these parameters to the procedure at run time, and will substitute the "$" references with the corresponding parameter. This makes the generic procedure into a specific procedure at the time it is executed. One procedure can be used for many purposes, which makes the programs smaller and much easier to maintain.

The last example also makes good use of Upper and lower case to make user defined words stand out. Procedures that are stored in the Procedure Library are not restricted to names with only 8 characters, so the longer "Pulse_Timer" name is not only legal but it is very descriptive.

These examples show all of the "style" issues that should be considered when writing SCADA programs. Obviously, it takes more original effort to do the "best" example rather than the first one. It takes more thought, and a bit more typing. However, the third example can be used a number of times, and is much easier to work with. In the end, a lot of time can be saved by the original programmer as well as any later ones when working with the more detailed example.

SYSTEM CONFIGURATION

The SCADA software is a complex multi-tasking program that runs under the DOS operating system on any type of IBM compatible PC computer. The program must be configured for each location so that the proper computer resources will be allocated in a manner that best fits the situation. For example, the number of tasks required for each system must be defined in an overall system setup file which is processed one time as the program initially loads. This file will tell the program how many tasks, their type, and how much CPU time and memory to allocate for each one. Although the setup is similar from one location to another, slight variations are required to suit the number of comm ports, local I/O drivers, and background calculation requirements.

The text file containing this configuration information is referred to as the main "DAT" file because the DOS file type defaults to ".DAT". The default file name is "RTU.DAT" and will be used if no other name is specified. However, the name of the file should reflect the location's identity, such as "VR167.DAT" or "MP36.DAT". The name is arbitrary, but should be selected with some care and should retain the ".DAT" filetype. This file will be the first thing processed after the program starts execution because it must do all of the memory allocation prior to loading any other system information. Any changes made to this file will require that the system is reloaded so that the file can be processed again. The file to use is specified on the DOS command line that starts the program with the /F= command line parameter as in:

```
RTUMON /F=VR167.DAT
```

The file name can contain any valid DOS path as well as the file name so that the configuration file can be on a different disk or directory than the main program. This is often done in ROM or RAM disk systems where the various system components are located on different logical devices.

The easiest way to build a new config file is to take an existing one and modify it. Modifications are made by simply typing over the existing text, or adding and deleting existing text. It is a god idea to "comment out" text that is not needed so that it will reman in the file as a reminder of an un-used option. Commenting is done with a semi-colon (;) character anywhere on the line, including at the first position. A comment character at the start of the line effectively deletes the line because it will not be processed. However, the line remains in the file and will be seen when the file is edited or printed as a note that a particular function was changed or disabled. Consider the following sample file:

```
password OFF
COM 1 1200 PHONE N 8 2 200 400
task,rtu,Com1 Phone, , 1
task,util, Utility RTU , ,4
task,Calc,Calculator, , 3
task,SCAN, Alarm Scan, ,4
task,driver,  ,32
task,Sec, One Second

driver metrabus ec311.mb   ; Request local I/O driver
```

```
LINKS 6
VARIABLES 8
name, EC311, EDC EC-311-AJ RTU  ; Complete the System name


rtu, EC311, EDC EC-311-AJ RTU,1
status  16
output   8
ain     16
timer   10
value    16
counter  2
aga3     3  ; need one for the faked meter
total    2
```

This example file contains an sample of every type of configuration line available. It is unusual to need all of the lines at every location, but it is possible. The syntax and purpose of these lines are detailed in the system design documentation, and will not be explained here.

TASK DEFINITION

Task 0 is always the local user, and is automatically defined by the system at load time. The user has limited control over this task as its memory allocation is determined by the program itself. However, the task manager settings can always be changed during program execution with the TASK command so that the task priority, delay, and other runtime setting can be manipulated.

Task 1 should always be the primary communications task for that system, or it should be the RADIO comm task if the system supports more than one comm channel. What this means is that systems with only a radio or phone should put the comm task as task 1, which will use COM1 as the serial port. If the system has both phone and radio, then put the RADIO on task 1 and the phone on Task 2. There is no technical reason for this, but it is a recommended practice to keep most systems the same.

Define the comm tasks one after another, and then define a UTILITY task that is needed at most systems. This is very similar to a comm task, but it receives its input via inter-task messages rather than from a comm port. Many system function require the presence of a UTIL task, so always put one unless there is a good reason to do otherwise. There should be only one UTIL task defined to avoid confusion.

After the UTIL task, we can define a CALCULATOR task that is very similar. The tasks are set up exactly the same, and differ only in their intended purpose. The UTIL task will process dynamic activities that can come at any time, and execute quickly. An example would be the processing of a report request or a daily log-off activity. The procedure runs once, after which the UTIL task is free to execute another procedure. The CALC task is normally used to process a continuous calculation procedure that runs and then repeats in an endless loop.

Once started, the CALC task is not available to process anything else because it is stuck in the loop. The CALC task is used for background calculations like tank levels, flow total math, or other constant requirements. This is not the same as TOTALIZER or AGA3 channels which are done automatically by the system. Rather, this is for user defined calculations such as figuring tank volume from a DP cell input, or adding several tank volumes together to produce a total into a value channel.

After the CALC task, which will be the last command processing type task, the I/O Driver and Alarm Scanning tasks must be defined. The I/O driver task is only needed of the system will have any local I/O. This would include any MetraBus, Generic, or Relay driver systems. Definition of the Driver Task will require later definition of the Drivers to be loaded and also the file names of the driver control files. Definition of the Driver Task simply allocates memory space for the task and puts it in the Task poll at the place where it was defined.

The Scanner task will constantly look at all channels and check for out-of-tolerance conditions and other alarm status changes. It also does the AGA3 calculations on each pass. The processor time needed for this task depends on the number of channels, and should be a higher priority for an RTU than a HOST. This is because a real RTU has data that is constantly changing, while a Host generally gets new data infrequently as a result of an RTU poll. So, wasting CPU time checking channels that rarely change results in a slower system than would otherwise be available.

The next task is the One Second task, which is a special task in terms of its priority. This task is not part of the normal task scheduling, and therefore must be the last one defined. It gets a chance to run once each second, and takes priority over other tasks at that instant. The task runs to completion, and the system then returns to whatever task was interrupted. The One Sec task handles timers, totalizers, and other system time related functions. It is given a high priority in terms of its system tick allocation so that it can always run to completion.

That is all that is needed to set up the tasks. The system will automatically allocate another task called the IDLE task as the very last one on the list. This task is necessary for the multi-tasker as it needs a place to go at times when no other task is ready to run. You do not need to define it, but it will show up in the task list of all STATUS displays after the system is up and running.

After defining the tasks, supplemental information is needed to further configure the system. Place any DRIVER statements just after the last task definition to inform the program which I/O devices will be needed at this location. The Driver statement tells which type of I/O (Metrabus, Generic, etc.) is being used, and also specifies the file name that should be used by the driver startup. This is a separate file from the one being considered here, and it will be processed much later in the startup process. There is a separate file for each driver type, and it tells the I/O driver the number of points and other driver specific information. Just keep in mind that all we do here is inform the system that the driver will be needed, and tell it where to find the driver configuration information when it needs it.

After all of the tasks and drivers are set up, we need to define the logical RTU systems at this location. An RTU is just a collection of I/O points, and there

can be many logical RTUs on a single system. Internally, the system will have a long list for each channel type. This list will contain all the information about the channels available at that computer. The list is subdivided into logical RTUs that represent a grouping of some sort, normally based on physical location. So, each RTU needs to be defined so that the system knows its name and local or remote status. After starting an RTU definition, we must specify the number of points for each channel type that will be used for this RTU. With this information, the program can build the overall point list for each channel type, and will know the start and end position for each RTU.

Each RTU definition will have a similar section, with the names and channel points being different. Each RTU does not have to have every available channel type. If there are no instances of a particular channel type, then it does not need to be mentioned. However, it is better to "comment out" the channel definition rather than delete it so that it is clear that no channels of that type are being defined for that RTU.

TASK 0 STARTUP FILE

After the system reads in all of the configuration files, it is ready to kick off the multi-tasking operating system built into the program. This is a very complex activity because the relationships among the tasks will vary from one system to another. However, there is some consistency in all systems so that they can follow this general startup guide.

The only thing automatically done by the program is that it makes the local task, TASK 0, ready to run and begins running it from the top. One consequence of starting any RTU type task (but not a Driver or other task) is that the task is instructed to read a file called "STARTx.RTU", where the x is replaced with the task's number. So, task 0 will begin execution with the message "READ START0" in its in-basket (so to speak!). So, the first thing task 0 will do is begin processing commands in START0, which will be the main kick-off for all other processes. No other task starts automatically, so it is up to task o, by reading START0, to get everything else going in the proper sequence.

The START0.RTU file must be a real DOS file and not a library function because the library has not been loaded yet. The only way to load it is with START0, so we have a chicken-and-egg problem. As a way of standardizing the overall startup process, the following generic START0 file can be used to load a library and then branch to the rest of the starting operation:

```
; Generic START0.RTU DOS file that gets it all going
library load $S.LIB
read Startup      ; Branch to the rest of the real startup process
```

With this method, the bulk of the real START0 operation would be contained in the library procedure called STARTUP. This way, START0 can be a small, simple DOS file that is the same for all locations, and the real STARTUP process can be embedded within the library. The end result is the same in that the computer must process the list of initialization commands normally associated with

the STARTO function. The change here is that the actual commands are placed in another procedure, called STARTUP, that is placed in the library. This is an important concept, and will be mentioned again later on.

This task 0 startup sequence is fairly standardized in concept, although the actual command lines will change to suit the location. The basic steps are as follows:

1. Read in the RTU configuration files for all RTUs assigned to this system. This should be done via GOSUB commands to each RTU, so they will be loaded right then one after another. A typical sequence is as follows:

```
Gosub Vr167 ; Load in RTU 1
gosub VR162
gosub WC240
```

This sequence would cause processing to branch to files called VR167.RTU, VR162.RTU, and WC240.RTU. These should be normal RTU files that should not be placed in the procedure library. This is because we want to dynamically change these files, and placing them in the library prevents this.

2. The next step is to load the comm link date that is stored in a file ending in ".LIN". This file is created every time a LINK SAVE is executed, and the name of the file is the same as the system name (determined in the main DAT file at load time). This name is available as a system parameter with $S, so it is possible to instruct the system to load the link file without even knowing its name. This is done as follows:

```
Gosub $S.lin
```

The $S will be replaced with the name of the system when the line is processed, so always do it this way and you will never have a problem. The only reason not to do this would be on a system with multiple link setting saved in separate files. These files would have user determined names rather than the system name, so it would be necessary to indicate the exact name on the gosub line. This is rare, so keep it simple and use the $S specification.

3. Load the desired agenda activities from a file called AGENDA.RTU. This is a special command file, and is read like any other command file with a GOSUB command. So, a typical line would be:

```
Gosub agenda   ; read in the daily activity list
```

4. Next, we can load in the "image" file that is generated automatically by the system every now and then. This file contains all sorts of dynamic setting related to alarm status, point values, and other system internals. Loading the image basically restores the system to the state it state it was in when the image file was last saved. On some systems, this is done every few seconds, so the value of timers, counters, totalizers, link states, and other

dynamic data will all be restored. This includes output settings, which may have to be overridden in the next step. Do not start the image save process just yet, as we may want to make a few more adjustments before we get that going. The command to load the image would be:

```
IMAGE LOAD
```

5. At this point, the default channel setting have been loaded from the individual config files, and then the last known system state was loaded with the IMAGE load line. No other tasks are running yet, so none of these values have any affect on system inputs or outputs. We have a last chance to force anything here before all of the automatic processes get going. The most likely thing is to force all outputs off to insure that they are not inadvertently activated when the driver task starts execution. So, a line like

```
CALC 01:o8 = off
```

could be used. This forces all outputs off in their memory image, and this is what the I/O device will see when it starts manipulating the outputs in real time.

Any other forced conditions could also be done here, such as forcing any channels into local or remote status, disabling any channels, or any other special tricks. Be sure to clearly comment each line so that the next guy will have some idea what is going on.

6. Each task can now be started, one after another, with some number of TASK command lines. These lines set the default priority and delay ticks, and then start the task. The exact order does usually matter, so simply go in numerical order. Remember that task 0 is already running (and is reading this START0 file), so it need not be re-started. A typical sequence could be:

```
Task 1 delay 4                  ; comm port task
Task 1 start
task Util delay 6               ;background tasker
task util start
task scan start                         ; alarm scanner
task driver delay 30                    ; Metrabus Driver
task driver start
task sec start          ; timer processor
```

Note that the tasks are referred to by their name, rather than their number. It is **BAD PRACTICE** to use numbers to refer to RTUs or TASKS because it is easy to mess things up if the RTU or TASK order is changed. Starting with the 03/03/91 version, selection of an RTU by number is no longer allowed. The RTU name must always be used. Tasks, however, can still be reference by number or name. The name of the comm tasks are just a number anyway, such as 1 or 2 for task 1 (typical radio) or task 2 (typical phone). The UTIL, CALC, SEC, SCAN, or DRIVER tasks should always be referenced using these names.

After this completes, all tasks will be set up to run and will participate in the

multi-tasker switching immediately. At this point, the system is in full operation.

7. Task 0 can now complete its startup by placing the local user in any state desired. This is normally at either the main system menu, or at a user programmable menu. If nothing else is done, then the user is left at the basic system command prompt.

8. Other RTU type tasks (i.e. those with comm ports, the Util task, and the CALC Task) also startup in much the same way as Task 0 in that they look for a STARTx.RTU file. Their needs are much less because they do not have to do all of the system startup procedures described above. They need only take care of their own needs, which is usually limited to setting the comm media (phone or radio), possibly setting the comm port number, baud rates, and other task specific parameters. Initial comm configuration command can also be done with the HAYES and PRC command lines that will configure a modem or PRC.

START FILE LOCATIONS

Only START0 needs to be in a DOS file, while all others can be in either DOS files of the procedure library. START0 needs to be in a separate file because at the time it is needed, no library could have been loaded yet. It is possible to put the bulk of the START0 file in the library with the following simple START0 file:

```
; Sample small START0 DOS file
library load $S.lib
read startup
```

With this trick, the small START0 file will load the main system library indicated by $S.LIB. If the system name is EI258, then the library file will be EI258.LIB. This file would contain most of the procedures necessary for system operation, including one called STARTUP. This procedure will contain the remainder of the Task 0 Startup sequence that would otherwise be located in the START0. The advantage of this is that all start0 files look the same because they do not reference any specific location. the $S does this for us at run time. This lets all of the procedures related to a location reside in one library file, which is easier to manage than lots of little files. The library also provides much faster execution because DOS is not used for procedure processing.

The only disadvantage of this is that the START0 file is fairly large, and it takes up space in the library buffer. For a small RTU, this is usually not a problem. However, a larger RTU or a HOST that has many procedures and Menu setups stored in the Library may need the space. Also, the START0 file can only be processed once at system startup, so there is little to gain by having it readily available in the Library.

Tasks other than Task 0 may be restarted from time to time, so it is a good idea to put their start files in the Library. This way, the start and bye files for each task can be grouped in the file, making for much easier procedure maintenance than when each procedure gets its own file.

NOTE: Some older program versions (before 08/91) had a bug that prevented proper location of STARTx files when they are in the library. In these cases, the START1 and START2 files must also be real DOS files and not library procedures.

CHANNEL SETUPS

Each I/O channel in the RTU system has many settings that control its action within the system. In it very likely that most channels will have slightly different settings from the others, although a group of channels often have similar settings. For example, all of the status input channels may often be specified as alarm type channels, while only some of the analogs usually generate alarms. Incorrectly setting a channel's alarm or callout settings will prevent the system from reacting in the desired manner.

All channel setups can be easily done using the CONFIG command or main menu pick. The configuration information for each logical RTU's channels are stored to disk with the SAVE command, and each RTU gets its own file. This simplifies movement of RTU config data from one system to another, but requires that the operator keep track of which RTUs have had changes so that the proper SAVE command will be executed. It is possible to directly edit the config file because it is stored in plain text as complex RTU command lines. These lines are normally processed during system startup, but can be processed at other times if required . The file name is the same as the RTU name, with a file type of ".RTU". This is the same file type as a normal RTU command file because the RTU config file is simply a special form of a command file. It contains valid commands that are fairly complex and would be awkward to enter by hand, although this is possible. The format of the command lines is complicated because it represents the three level channel definition used by the system. So, each channel may have up to 3 lines of definition associated with it. Again, it is rarely necessary to directly edit this file, because the config screen makes channel setup much easier.

It is a good idea to place some sort of time delay on each channel that will defer its alarm actions for the specified number of seconds. Even a delay of 1 or 2 seconds can be used to eliminate many false alarms. This is because it is not uncommon to have momentary conditions that appear to be alarms but are actually caused by some otherwise harmless activity. One example is the manual reset of a control panel which often activates circuits that indicate failure as part of the reset process. IF an alarm is being detected by the voltage to an alarm light, then pressing the Lamp Test button for a few seconds would incorrectly signal the RTU that an alarm had occurred.

Other points may require fairly long delays to prevent false alarms. This is the case with pressure and temperature type inputs that may have momentary high or low conditions in the course of normal activity. A pipeline pressure may rise or fall as a result of actions at another facility, and it may take a minute or so for the pressure to stabilize after the abrupt change. Battery voltages may dip fairly low in the middle of a cold night, so a delay of several hours may be needed there to provide a more accurate alarm condition.

Obviously, setting these delays requires some knowledge of the process being monitored. The delays can be easily changed at the RTU, or they can be changed at the HOST location and sent down to the RTU over the comm line. So,

always implement some sort of delay at system installation and fine tune it later after the system has had time to perform under actual conditions.

ALARM DEADBANDS

The deadband feature of all value derivative channels is a great feature that will assist in eliminating many repeated alarms from a channel hovering near its setpoint. The deadband applies to both the high and low setting, and will determine the channel value that will allow a return to normal after an alarm sequence. For example, a battery monitor with a low setpoint of 11 volts and a deadband of 1.5 volts will go into low alarm at exactly 11 volts, but will remain in alarm until the voltage rises above 12.5 (11 + 1.5). So, as the battery hangs around 11 volts, say going from 11 to 11.2 , back down to 10.9, back up to 11.3, and so on, it will not generate a fresh alarm each time it passes through 11 volts.

Deadbands, like alarm delays, may have to be estimated during installation and fine tuned after some period of service. They should be present on every value type channel that is set up as an alarm channel.

CALL ON ALARM AND RESET

This setting is present for each channel to help control communications costs for systems with Cellular Phones, toll calls, or limited power supplies. The cost of the call (in dollars and electrical power) must be assessed in relation to each point. If communications are basically free, then set each channel to call on both alarm and reset. This will eliminate any confusion or out-of-sync problems between the RTU and HOST. However, if calls are expensive, then some determination must be made as to which points should be allowed to generate calls.

This is not as difficult as it could be because most systems have alarms that do not occur very often, so having every alarm point generate calls is normally not a problem. Unless there is some reason to do otherwise, set all alarm points to call on alarm as well as reset. If a point becomes a problem, it can always be changed later. This is better than missing alarms.

CHANNEL NAMES, UNITS, AND ALARM PHRASES

These items are the text description assigned to each individual point, either to describe the point itself or its current status. The text is arbitrary, and should be selected to suit the user and the installation requirements. Often, the use of abbreviations helps reduce screen clutter and makes the system easier to use. Also, the use of lower case for normal phrases and UPPER CASE for alarm phrases may make for a more pleasing display. In no case should everything be typed in upper case, as it makes the screens far to cluttered and makes it more difficult to locate points that are in alarm.

For status type channels, the normal and abnormal phrases should make sense for the channel description. For example, the text definition for a platform shutin alarm could be "Well #1 Status", or it could be "Well #1 Alarm". These are

slightly different phrases, and would require different styles of alarm phrases. With this "Status" description, the alarm status phrases could be "Online" and "SHUTIN". With the "Alarm" description, the phrases would be "Off" and "ACTIVE". A poor choice would be "Off"and "On". The user could be confused by "Off", which means that the alarm is off, by thinking it means the well is off. Do not confuse the process status with the alarm status. Most applications seem to work better with a process status, and let the phrases indicate the status. It is more confusing to name a point after the alarm status, and let the phrases reflect the alarm state rather than the process state.

The choice is up to the user, but a good tip is to be consistent at each location, and also within each company. If you use "Separator Status" for one point, use "Heater Status" for a similar point. It may seem like a small point now, but it really helps the daily operators who are not very familiar with SCADA systems if these text descriptions and phrases use the same language they do.

The Units for Value type channels can follow the same guides as the text descriptions. If the operators use MMCF to indicate Millions of Cubic Feet per day, then do not put "MIL/DAY" in the units field. Most industries have very set ways of indicating values, and these same units should be used on the SCADA system. The use of upper and lower case can be a help here, as "Bbls" is more common than "BBLS" to describe barrels. Again, the exact format is up to the user, but be consistent within the same system and company.

OUTPUTS AND TIMER CHANNELS

The main use of timer channels at an RTU is to pulse output channels. Timers are also used for a few other functions, but output pulsing is the main event. So, it makes a lot of sense to set up the timers in the EXACT same order as the output channels, even if it means putting in a few unnecessary timers that will never be used. Timer channels are free, so it is hard to waste them. If possible, lay out the outputs such that they are grouped is a logical manner, and that the ones needing timers are at the top of the output list. This way, a shorter list of timers can be used that will line up one-for-one with the output channels.

If you have to put in a few dummy timers that will not actually be linked with an output, you can always use them for other timer functions like call-back or hurricane timers. Name the timer channels so that they relate to the output channel. For example, "ESD Pulse Timer" is a lot better than "Output 1 Timer".

AUTOMATIC ACTION PROCEDURES AND FILES

CONNECT FILES

Each comm task starts communications when the carrier detect (CD) line on the serial port becomes active. The task goes from an offline (standby) to a online (active) state and expects commands to come into the serial port. It will send result messages out of the serial port. While online, numerous error checking and timeout procedures go into effect to prevent the task from getting hung up with screwy comm links.

When the task first goes online, it will look for a special file called CONNECTx.RTU, where the x is replaced with the task number. So, task 1 will look for CONNECT1.RTU. These files were important in early versions of the software (circa 1988) because they controlled the communications session. After the introduction of the LINK system, the CONNECT files were mostly obsolete. However, they were retained for cases where some sort of special processing is needed at the start of a comm session. they are no longer used to control the actual data transfers, but are still used for any hardware control necessary to manipulate the comm equipment. For example, if it were necessary to activate an output when the comm session started, a file like:

```
; comm starting, activate aux battery.
calc o5 = on
```

could be used. A corresponding output turnoff command would probably be used in the BYE file described below.

LINK FILES

The LINK files are processed by a task whenever it is originating a call due to an active Link. Incoming calls do not use the Link system at all. The purpose of this file is to allow special processing when a task performs a callout. Normally, the task will use it's HOST or RTU status setting for the current link in order to determine what to do when a callout makes a connection. The RTU status causes the task to process a file called DOWNLOAD.RTU, which normally results in the RTU sending channel data to the answering unit. A HOST status causes the task to send a message to the other computer telling it to process its DOWNLOAD.RTU file, resulting in a data transfer FROM the other unit. So, RTUs normally send data on callout, while HOSTS request data on a callout.

There are times when this is not the desired action, such as when a HOST calls another HOST. In these cases, the active link for the calling HOST will need a control file telling it what to do when it connects on a callout. That action may be to send data, receive data, or both, depending on the situation. If a LINKx file is present, it will be processed. If it is not there, the RTU or HOST setting will control the action.

Note that the x in the LINKx.RTU file name refers to the Link number, not the task number as was done in the other file types. So, each link (representing a different phone number and probably a different unit) can have its own unique special process that will occur when a callout is complete.

DOWNLOAD FILES

The standard command file that will transfer data from a unit is called DOWNLOAD.RTU. This file will be unique to each location, although all are very similar. The basic steps are as follows:

1. Set ONLINE ON to prevent processing of the file if the comm link is lost. The ONLINE flag remains active until turned off or until the command file

completes execution.

2. Send channel data using SCAN lines for all channel types to be sent to the other unit. The last line should contain the special @ sign to indicate that it is the last line. This will cause the other unit to timestamp the data with its local time. It also causes a current link at either end to be reset.

3. ACK and RESET the local channels so that their alarm state will move from NEW to IN-ALARM, or from IN-ALARM to NORMAL. This is important in order to allow a point to clear and then be able to re-alarm when needed.

4. At this point, the SET ONLINE can be turned off because a loss of comm will not cause a problem. We have already sent the data and cleared the local alarms. Any new alarms should be active at the other end by now.

5. Terminate the communications by doing a BYE command. This is normally at the end of the DOWNLOAD.RTU file.

A typical file may look like this:

```
; Typical Download file for a small RTU
set online on
scan s1:s16 R o1:o8 R a1:a8 E
scan q1:q2 E q1:q2 T c1:c4 E c1:c4 T
scan m1:m2 E v1:v8 E@
reset
ack
set online off
bye   ; end of download file
```

BYE FILES

Each comm task will look for and process a BYEx.RTU file each time a communications session ends. This is triggered by the loss of carrier detect (CD) on the serial port. This file is used to do whatever is necessary to disconnect the comm link and re-program the modem or PRC to insure it is properly configured. This file also runs after long periods of no communications as determined by the SET COMM command. This is normally set to several hours so that any comm inactivity that resulted from an improperly configured modem will be cleared automatically.

AGENDA SETUP FILES

The program's agenda system is a special background function that watches the clock and issues commands to various tasks at specified times of day. This is used to schedule special events such as production logs, RTU polls, or other daily routine tasks. The AGENDA command can be used from the command prompt for manual entry, but this is awkward and rarely used except for quick tests. The correct way to set up the agenda is to place all of the desired agenda commands in a separate file, surprisingly called AGENDA.RTU. This file, which should not be placed in the library, contains the list of things to do. It is normally read during

initial startup, but can be edited and re-read at any time. Placing all of the commands in a single file centralizes the list and makes management easier.

Each agenda line consists of the keyword AGENDA, a task to receive the message, a time of day, and a command message. This information is stored in a memory list and is sent to the specified task at the appropriate time. It is best to make the agenda commands very short, and to place complex lists of commands in separate command files or library procedures. This way, the agenda is simple, while the intended function may be very complex.

For example, many RTUs perform some sort of daily production log procedure early in the morning. This consists of saving some previous day values, clearing the counters and totalizers, and possibly doing a little math. All of this may take 20 program lines to complete the function. All 20 lines could be put directly in the agenda, but it is easier to place the lines in a file or procedure called DAILY, and simply tell the agenda to READ DAILY. This way, the agenda is not cluttered and maintenance of the daily function is centralized in one command file.

Using this scheme, the agenda activity should consist of a list of "one liners" such as READ DAILY, LINK ALL RESET, POLL ALL NOW, or other functions that can be done in a single line. Anything more should be placed in a separate file.

The agenda is usually loaded with a GOSUB AGENDA line in the STARTO file, which tells the program to branch to a file or procedure called AGENDA. After file processing is complete, the startup will continue with the next line. The agenda requires several control lines to clear and then activate it whenever a new list is loaded. So, the AGENDA file will normally start with an AGENDA CLEAR command, followed by the list of agenda commands, and ending with an AGENDA ON line. This makes the agenda file a self contained unit. At any time, the operator could edit the AGENDA file, and then issue a READ AGENDA command to have the system clear, reload, and reactivate the agenda list.

A typical AGENDA.RTU file is as follows:

```
; Typical Agenda file
agenda clear  ; remove any existing agenda activity
agenda,util,02:00:00,link all retry ; retry all failed links
agenda,util,06:00:00,Read Daily
agenda,util,07:00:00, poll all now
agenda,1,23:00:00,bye  ; Make sure com1 is reset every night
agenda on             ; start the background process
```

Note that all agenda entries must be in time of day order, from early to late. An out of sequence entry will be executed at the wrong time.

After loading, simply enter the command AGENDA to have the system dump the list for your review and approval. If anything is wrong, edit the file and check for misspelled words or an out of sequence entry.

MENU SETUPS

Menu files are different from normal RTU command files, but they are built using the same editor and common sense rules as the other files discussed here. The philosophy behind the menu structure will have to vary among systems because each one is unique. However, there are some common features we want to see in all the menu systems on all the RTUs.

The main decision to be made is now to generally divide the various menu functions. For example, it may be possible to separate the menus into function groups, or they may be separated by physical RTU location. In other words, we could have menus that have titles like "Display Menu", "Shut-In Menu", and "Utility Menu". These menus would have similar features for several RTUs on a single menu. An alternate method would be to use separate menus for each location, with a similar list of functions on each one. These menus would be titled "RTU 1 Menu", "RTU 2 Menu", etc. The choice is up to the operator, but be consistent at different locations for the same owner.

The menus should be "nested", where a main menu "calls" a lower level one, and the lower one "returns" back to the one that called it. This is done with the MENU GOSUB and MENU RETURN commands. Older versions of the program could only load a new menu with no way to go back to a previous menu without coding in the exact name of the menu file. Now, a menu can be loaded under an existing one, and the lower one can go back to the previous menu without knowing anything about the menu. Menus can be nested up to 8 deep, which is more than adequate for even the largest Host system.

The Menu titles should be clear, and not wordy. The title "VR-256 Main Menu" is fine, while "TMC VR-256-A MAIN RTU COMPUTER MENU" is way to long. Also, the use of Upper case for emphasis is highly recommended. TO MANY UPPER CASE LETTERS LOOKS BAD AND IS HARDER TO READ. To Many UPPER Case letters look BAD and are HARD to read. See what I mean?

The first entry on every menu should be "Return to Previous Menu". This provides a consistent way for a user to navigate the menu structure. The top entry always takes him back one step. Avoid menu entries that allow the user to jump sideways across the menu structure,

SHIFTED FUNCTION KEYS

The 03/91 program version introduced a user programmable function key feature that allows each location to have 9 "hot keys" that are accessed with the Shifted Function Keys. Shift-F7 is already used by the program for "Display First", but the other keys are open for grabs. Unless these keys are needed for a special purpose, the following list is a good use for these keys:

SF1 Turn on Comm Watches (1, 2 or both)
SF2 Turn off the Comm Watches
SF8 Re-Load the Procedure Library
SF9 Go directly to USER menu
SF10 Put up a short help file.

These functions can be placed in the standard library file and easily used on each rtu. The function keys can be used for other purposes, but use this list unless there is a good reason not to.

EXAMPLE PROCEDURE LIBRARY

The following example was taken from an actual installation, with some modifications just for illustration purposes. It is provided here in full form, and was from a Host Computer that monitored information from 3 RTU locations. This file can be used as a guide when setting up other systems.

```
;=============================================================
;library for moxy wc 458 host   AMZ  05/12/91

PROC Shut0   ; Task 0 ask it OK to perform timer pulse action

  pause,Do you want to continue?,n,120
  force 1 read shut1 $1 $2
  force clear 0


PROC Shut1   ; Task 1 to pulse an output for any RTU.
               ; Param 1 is RTU, 2 is timer, 3 is seconds
  sleep 1   ; Wait a sec for Task 0 screen to get cleared
  sele $1
  msg Dialing RTU $r
  dial
  wait 30 conn
  if comstat(1)
    msg RTU $R is online now.
  else
    msg RTU did not connect.  Retry later.
  endif
  set online on

  msg Rtu $R is online.  Sending Command to pulse solenoid
  BLOCK calc $2=$3
  msg Command Sent.  Rtu should be Pulsing Solenoid now.
  msg Waiting 10 seconds for download
  sleep 5
  msg Waiting a bit more.....
  sleep 5
  msg Getting Download
  block read download
  force 0 disp s1         ; Puts local user into display

PROC Change0    ; Ask if OK to send values to an RTU
  cls
  Echo This will Call $1 and upload the Analog Setpoints and Value Channels
  cursor 1,8
  pause,Do you want to continue?,n,120
  force 1 read change1 $1
```

```
  force clear 0


PROC change1   ; Task 1 proc to call RTu end send values, setpoints
  sele $1
  dial
  msg Calling $R to send values and setpoints
  wait 30 conn
  set online on
  msg RTU is online.  Sending Analog Setpoints
  scan a1:a16 l
  scan a1:a16 h
  scan a1:a16 d
  msg Sending Value Channels
  scan v1:v16 E
  scan s1:s16 w a1:a16 w
  msg Telling RTU to save its new setup
  block save
  msg Signing off
  set online off
  bye


PROC STARTUP   ; completion of the system startup.
                        ; Called from Start0 Dos File at startup time.
  set print on
  cls
  msg Loading WC433
  gosub wc433a.rtu
  msg Loading WC457
  gosub wc457b.rtu
  msg Loading EC311
  gosub ec311.rtu
  msg Loading WC458
  gosub wc458a.rtu
  msg Loading WC459
  gosub wc459.rtu
  msg Loading Agenda List
  gosub agenda
  gosub $S.lin

  task 1 start
  task scan  start
  task util start
  task driver start
  task sec start
  sele wc458a
  SET HORN BUZZ
  set warb on
  LOG EVERY 1
  image load
  image on 20
```

```
PROC alog   ; :List the alarm log

  SELE $1
  CLS
  LOG
  FORCE CLEAR 0

PROC hurc0   ; task 0 hurricane prompt

  sele ec311
  cls
  msg This procedure will set a timer at EC-311 that will shuting the facility
  msg when it reaches 0.  The action can be cancelled by setting the timer to
  msg 0 rather than some number of hours.
  cursor 1,8
  pause,Do you want to continue?,n,120
  cursor ,12
  input,Enter number of hours,t9
  force 1 read hurc_1
  force clear 0

PROC hurc_1   ; Call EC311 and set the hurricane timer

  stuff 0 esc esc esc esc esc esc
  sele ec311
  msg Calling $R to set Hurricane Timer
  dial
  wait 30 conn
  set online on
  msg Setting timer T9 at EC311
  scan t9:t9
  msg Timer set.  Disconnecting now.
  set online off
  bye


PROC Pollall; activate the link for all rtus

  sele ec311
  poll now
  sele wc459
  poll now
  sele wc457b
  poll now
  sele wc433a
  poll now


PROC Start1  ; standard radio startup
  set media radio
  prc mycall WC458a
  msg Task 1 started at $T
```

```
PROC Bye1   ; standard radio disconnect
  Set Echo Off
  prc restart
  prc status off
  prc echo off
  prc monitor off
  prc mycall WC458A
  prc xflow off
  prc flow off
  prc txdelay 70


PROC Disp_Ec    ; custom display for ec311
  sele ec311
  cls
  if $1=0
  box 1,1,65,24
  cursor 18,20,"HIT ESC TO TERMINATE DISPLAY
  endif
  :loop
  cursor off
  cursor 2,5,"McMoRan EC-311-AJ PRODUCTION DATA Updated - $@
  cursor 12,8,"Sales Meter No. 1`l20,M1," MMCF/Day
  cursor 12,9,"Sales Meter No. 2`l20,m2," MMCF/Day
  cursor 37,10,"------
  cursor 12,11,"Total Gas Rate  `l20,(m1+m2)," MMCF/Day
  cursor 12,13,"Cond. Sales No.1 `l20,c1," Bbls
  cursor 12,14,"Cond. Sales No.2 `l20,c2," Bbls
  cursor 37,15,"_______
  cursor 12,17,"Total Cond. Sales`l20,(c1+c2)," Bbls
  if $1=1
    dump screen
    cls
    cursor on
    return
    endif
  if @key(0) > 0
    cls
    cursor on
    return
  endif
  goto loop

; Manual meter calculation
PROC Man_Calc
  cls
  msg This procedure uses a faked AGA3 meter at WC-458 to allow manual entry
  msg of the various parameters to simulate a real flow meter.
  msg You must verify the pipe size and enter the flow information.

  cursor 1,10
  pause,Ok to continue,Y,120
  msg Setting up to configure Faked Meter
```

```
  sele wc458a
  config m1
  msg Now you have to verify or change the other parameters.
  msg The prompt will show you the current value.  Either enter a new value
  msg or simply hit [ENTER] to leave it as it is.

  cursor 1,8
  input,Enter Plate Size, f_plate
  input,Enter Pressure,f_static
  input,Enter Delta-P,f_diff
  input,Enter Temperature,f_temp
  input,Enter Specific Gravity,f_sg
  input,Enter Percent N2,f_Mn2
  input,Enter Percent CO2,f_co2
  aga3 m1
  disp m


PROC   SF1   ; shift key F1
  cls
  msg Turning On Com 1 Watch
  watch 1 on

PROC SF2
  msg Turning Off Com 1 Watch
  watch 1 off

proc sf5        ; a super F5
    read pollall

PROC SF8   reload the library
  force 0 lib clear
  force 0 lib load wc458a.lib

PROC sf9   ; user menu
  user

PROC SF10
  cls
  help

;--------------  COMM FAILREUS

PROC comfail1  ; set comm failure alarm
  read alert comfail1

PROC comfail2  ; set comm failure alarm
  read alert comfail2

PROC comfail3  ; set comm failure alarm
  read alert comfail3

PROC comfail4 ; set comm failure alarm
```

```
  read alert comfail4

PROC alert  ; set comm failure alarm;  Param 1 is the channel
  sele wc458a
  calc $1 = 0
  sleep 1
  calc $1 = 1

PROC clear_alert   ; Remove the comm failure alarms
  sele wc458a
  calc s1:s4 = 0

;   ------------------- M E N U S -------------------

PROC Utility       ; Utiulity menu
`UTILITY MENU " 20 " 5
|BACK PREVIOUS MENU
  menu return
|Clear Com Fail alarms
  gosub clear_alert
|Edit Agenda List
  edit agenda.rtu
|Reload Agenda List
  read agenda
|Edit Procedure Library
  edit wc458a.lib
|Reload Procedure Library
  read libload
|DISABLE PRINTER
  set printer off
|ENABLE PRINTER
  set printer on


PROC ec311m
`EC-311-AJ RTU OPTION MENU" 20 " 5
|RETURN TO PREVIOUS MENU
  menu return
|Show Production Display
  read disp_ec 0
|Print Production Display
  read disp_ec 1
|Send Setpoints and Values
  read change0 EC311
|EC-311 Shut-In Menu
  menu gosub ec311M2
|See alarm Log
  READ alog ec311
|AGA3 ERROR CODES
  type aga3
  force clear 0
```

```
PROC EC311M2   ; Menu
`EC-311-AJ RTU SHUTIN MENU` 20 ~ 1
|RETURN TO PREVIOUS MENU
  menu return
|Platform ESD
  MSG This will cause a platform ESD
  read shut0 ec311 t1
|Platform Total S/D
  msg This will cause a Total S/D
  read shut0 ec311 t2
|Shut In All Wells
  msg This will shutin all the wells
  read shut0 ec311 t3:t8
|Shut In Well A-1
  msg This will shutin well A-1
  read shut0 ec311 t3
|Shut in Well A-2
  msg This will shutin well A-2
  read shut0 ec311 t4
|Shut In Well A-3
  msg This will cause well A-3 to shutin
  read shut0 ec311 t5
|Shut in Well A-4
  msg This will cause well A-4 to shutin
  read shut0 ec311 t6
|Shut in Well A-5
  msg This will cause well A-5 to shutin
  read shut0 ec311 t7
|Shut in Well A-6
  msg This will cause well A-6 to shutin
  read shut0 ec311 t8
|Shut In Platform ESD
  msg This will cause a platform ESD
  read shut0 ec311 t1
|Set Hurricane Timer
  msg This will start the hurricane timer
  read hurc0

PROC WC433AM
`WC-433-A OPTIONS MENU
|RETURN TO PREVIOUS MENU
  menu return
|Send Setpoints to Wc433
  msg This will send the setpoints in your computer
  READ CHANGED WC433A
|Shut In WC-433 A
  msg This will cause a well panel shutdown
  read shut0 wc433a t1
|Shut In Well A-1
  msg This will cause well A-1 to shutin
  read shut0 wc433a t2
|Reset  Well A-1
  msg This will reset well A-1
```

```
  read shut0 wc433a t3
|Shut-In Well A-2
  msg This will cause well A-2 to shutin
  read shut0 wc433a t4
|Reset   Well A-2
  msg This will reset well A-2
  read shut0 wc433a t5
|See Alarm Log
  READ alog wc433a


PROC WC457BM
'WC-457-B OPTIONS MENU " 30 " 1
|RETURN TO PREVIOUS MENU
  menu return
|Send Setpoints to WC-457B
  msg This will send the setpoints in your computer
  read change0 wc457b
|Shut In WC-457 B
  msg This will cause a platform ESD at WC457b
  read shut0 wc457b t1
|Shut-In WELL B-1
  msg This will cause well B-1 to shutin
  read shut0 wc457b t2
|Reset   Well B-1
  msg This will reset well B-1
  read shut0 wc457b t3
|Shut-In Well B-2
  msg This will cause well B-2 to shutin
  read shut0 wc457b t4
|Reset   Well B-2
  msg This will reset well B-2
  read shut0 wc457b t5
|See Alarm LOG
  READ alog wc457b


PROC WC458M
'WC-458-A OPTIONS MENU " 30 " 1
|Return to Previous Menu
  menu return
|Nothing on This Menu yet
  menu return


PROC WC459M
'WC-459 OPTIONS MENU " 30 " 1
|Return to Previous Menu
  menu return
|Send Setpoints to WC-459
  MSG This will send the setpoints to the RTU
  read change0 wc459
|Shut In  C-459 ESD
```

```
  msg This will cause a platform ESD
  read shut0 wc459 t1
|Shut-In Well CJ-1
  msg This will cause well CJ-1 to shutin
  read shut0 wc459 t2
|Reset   Well CJ-1
  msg This will reset well CJ-1
  read shut0 wc459 t3
```