

# System Extensions

*Can you do alterations?*

Oct 6, 1989

The Local Station system software has evolved over a number of years as new features have been added and changed. This document describes procedures that may be used to make some commonly-requested additions.

## *Add a new application page*

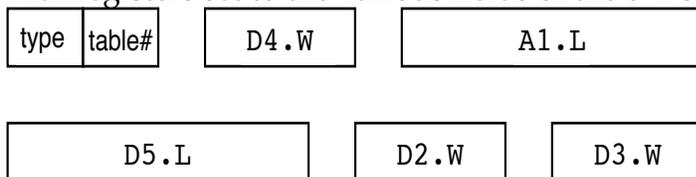
This is done without any system code changes. After preparing the program in S-record format using the cross compiler and/or assembler, find an area in non-volatile memory (so the application will survive a power down) sufficient to contain the linked application. Download the code into that area using the Download Page to process the S-records sent to a serial port of one local station. Go to the index page and find an available page slot to be used in calling up the new application. Invoke the list of entry points of the applications associated with each page by calling up the index page again with the hex switch (one of the small console units buttons) depressed. Enter the starting address—which is assumed to be the entry point—of the application area and press the interrupt button with the cursor just beyond the last character of the 8-digit address *with the hex switch depressed*. (Obviously, without the hex switch depressed, that page would be invoked at the old entry address.) At this point, you should notice that the newly-entered address is displayed as it was typed, and there is no “-” in the second character position of the line. If there is a “-” present, it means that the entry point address does not seem to be valid, and the system will refuse to invoke it. It must be even and not too small, and it must point to a word of memory with the value 0x47FA, the opword for a LEA disp(A3) instruction, which must be the first instruction of every application page program. Then call up the new application as usual.

To provide a title to the application, call up the page, type the 16-character title on the top line starting in the third character position, and return to the index page by placing the cursor in the home position and interrupting. The new title will be available on the index page and at the top of the application page the next time it is invoked.

## *Add a new Data Access Table entry type*

Design a 16-byte entry format to be used for the new type. The first byte is the type#, a small positive value chosen by checking the branch table READS at the end of the RDADNEW module. The second byte is assumed to be a destination table#, which is usually 0x00 to denote the ADATA table for analog channel readings or 0x05 to denote the BBYTE table for binary byte data readings. If no table# is required, set it to a negative value (like 0xFF) to denote that it is not a table# to get around the check for the entry# being out of range for the table size. (The auto-setting entry type uses this.) The next two bytes (the second word) are assumed to be the destination table entry#, and is therefore a Chan# (for the ADATA case) or a Byte# (for the BBYTE case).

Write a routine to process the entry and add its entry point to the branch table READS mentioned above. If the routine is external to the RDADNEW module, declare an XREF of course. The routine is called with registers set to the various fields of the 8-word entry as follows:



In addition,

- D1.L= offset to the destination table entry in the table
- D6.W= #bytes/entry in destination table

A2.L= ptr to destination table entry  
Condition codes set by TST.W D3 instruction

All registers may be altered by the routine *except* A3/A5/A6/A7. Examine other routines for examples. The document entitled “*RDATA Entry Formats*” describes the current entry types.

### **Add new Analog Control type**

The SETAC module contains many routines selected by the analog control type byte in the analog control field of the analog descriptor. That field is currently 4 bytes in length. The first byte is the type# byte, and the meaning of the other 3 bytes depends upon the type#. A setting to an analog channel device results in a call to one of these routines.

To add a new type of analog control routine, design a data structure that can be used for the analog control field:



Add a new routine reference to both the branch table SETACS and the branch table SETREL at the end of the SETAC module. Write the SETACS routine consistent with the following register-based calling sequence:

D4.W= dataword to be set  
A0.L= ptr to analog control field of analog descriptor for this channel  
A4.L= ptr to setting word in ADATA table for this channel

Any registers may be altered except A3/A5/A6/A7. By convention, the routine should include copying the data word into the setting word of the ADATA entry iff no errors are detected in processing the setting. In this way, a readback of the setting value (following the setting command) can determine whether a setting to an analog channel was successful.

To support knob relative settings, the SETREL branch table invokes a routine which scales the knob click, the dataword for the relative setting (listype=7) case, based upon the analog control type. (This may not always be sufficient; the case of 1553 analog control required a separate type# for 12-bit and 16-bit D/A relative control.) The scaled knob click value plus the setting word forms the intended setting value.

### **Add a new read type routine**

An entry in the Listype Table (module LTT) indexed by listype# includes a read type#. The routine indexed by this value is in the READS table at the end of the COLLECT module. (Don't be confused by the name READS also being used in the RDATA module; they are different branch tables.) It is invoked by an application program's call to Collect and also by the Update task when updating network requests. (The Server task also calls it using CollectS.)

The routine has a register-based calling sequence. Upon entry to the routine,

D0.W= #idents-1 (or #internal ptrs - 1, since there is one ptr per ident)  
D1.W= #bytes to return (>0)  
A1.L= Ptr to array of internal ptrs (corresponding to original array of idents)  
A2.L= Ptr to data array to be filled

The significance of an internal ptr depends on the code in the REQDGENP or PREQDGEN modules that generated the internal ptr. It is typically a ptr to an entry in a system table, or it may be a ptr into an external answer buffer, usually with the sign bit set to indicate this, or it may be a ptr to a

source of zeros—a null ptr. Whatever it is, the read type routine must be aware of its possibilities.

Upon exit from the read type routine, the A2 register must be advanced past the data area of answers produced in satisfying the array of idents. The calling routine then will “even up” the A2 address so that the answers for the next listype, if any, in the request will start on a word boundary. Note that an odd #bytes in a data request will only result in a filler byte after processing the array of idents. Normally, the only odd #bytes likely to be used is 1.

Also upon exit from the read type routine, if the condition code status indicates overflow, the calling routine will assume a bus error occurred during processing and will return an error code 4 to the user.

Besides the A2 register and the condition code status, all registers are available to the read type routine, except A3/A5/A6/A7.

Most of the current read type routines are found in the COLLECT module.

### *Add a new set type routine*

An entry in the Listype Table (module LTT) indexed by listype# includes a set type#. The routine indexed by this value is in the SETS table at the end of the SETDATA module. It is invoked by an application program’s call to SetData and also by the Network task when processing setting messages from the network. There are two variations of set type routines. The first is used if the ptr type byte is < 32, indicating a system table#. In this case, the ident is assumed to be a table entry# and is checked to be within the range of the table. The second variation is used if the ptr type byte is ≥ 32.

Upon entry to a set type routine,

- D2.w= #bytes of data
- D5.w= ptr info byte
- D6.w= 0 if short ident, -2 if long ident
- A1.L= ptr to data
- A2.L= ptr to ident

In addition, if the ptr type < 32, system table parameters are made available as

- D4.w= entry# from ident
- A4.L= ptr to field in table entry (using D5.w as offset to field in entry)

### *Add a new ptr type routine*

When adding a new read type routine, it is sometimes necessary to add a new ptr type routine as well. The ptr type routine generates an *internal ptr* from an ident. The read type routine generates answer data from an internal ptr. For more explanation of ptr type routines, see the document entitled “*Internal Ptrs.*”