Writing Auxiliary Processors for APL2

February 18th, 1988

Ray Trimble

IBM Corporation M46/B25
P.O. Box 49023
San Jose CA 95161-9023
Preface

APL2 has included, since Release 1, a rich set of system independent auxiliary processor services and interfaces. Customer documentation for these facilities is only now becoming generally available.

This paper will provide a survey of those facilities and an example of their use. Included are services to provide data conversion, error handling, file system access, message formatting, multi-tasking, shared variable processing, terminal control, and dynamic virtual storage.

Most of these same services are also available to Processor 11 Function Routines, and the material here should also be helpful in writing those. But the focus of this paper will be auxiliary processors.

Familiarity with the concepts of APL shared variables and auxiliary processors is assumed. Complete details of these facilities are provided in SH20-9234 APL2 Programming: Processor Interface Reference.

Considerable reference is made to the term "CDR." This is an acronym for Common Data Representation, and refers to an interchange data format which was formerly defined in Appendix A of SH20-9215 APL2 Migration Guide, but is now presented in the Processor Interface Reference.
Starting an Auxiliary Processor

APL2 supports two distinct types of auxiliary processors (APs) which are started in quite different ways.

- **Global APs** are system-wide servers. They share variables, often concurrently, with multiple APL sessions. These APs are typically started during operating system initialization. Under VM they execute in separate disconnected virtual machines. Under MVS they each execute in their own address space.

- **Local APs** are written to share with only one APL session. A separate instance is created for each user that wants to use a given local AP. The APs are started by the APL2 executor, normally during user invocation of APL2. They execute under control of the user's virtual machine or address space.

Global APs are given control directly by the operating system, and obtain most of their services from it. They will not be discussed further in this paper.

Local Auxiliary Processor Entry

When an auxiliary processor is started (by APL2), it is given control using a standard CALL linkage. APL2 provides a parameter list, as follows:

1. (Used only by VS APL compatibility support.)
2. A pointer to a service routine which may be called by the auxiliary processor. This service routine supports the services described later.
3. The beginning of a model parameter list for the Virtual storage service (see the VP service described later). This helps solve the bootstrapping problem for reentrant programs of needing storage for the request that obtains storage.
4. The second parameter for the VP service call.
5. The third parameter for the VP service call.
6. (Reserved)
7. The length of any string being passed to the AP by APL invocation.
8. The parameter string (if any) provided in the APNAMES invocation option.

Local Auxiliary Processor Exit

Auxiliary processors should terminate when they receive a CSVENA return code from an SVP service, or observe the "sign off" signal sent by the SVP. This signal will always be posted in the processor ECB. Processors should break their connection with the SVP before terminating.

On normal termination registers must be as at entry, and the processor must return to the address in register 14.

An abnormal termination will occur if an unrecovered program check or ABEND occurs in the auxiliary processor task. Processors may recover from all program checks and most abends by using the EX service described later.

---

1 Throughout this paper, numbered lists are used to represent parameters, by number, in a parameter list.
Data Conversion Services

There are several services in this group:

DE  Translate from VS APL Zcode to EBCDIC
DN  Change the data format of numbers
DU  Translate with user-supplied table
DX  Convert Extended Character data
DZ  Translate from EBCDIC to VS APL Zcode

Of these, probably only the DN service is of general interest.

DN: Change Data Format of One or More Numbers

This service produces a list of numbers in the output area, in the format specified by the output type. The input area is analyzed according to the input type. The caller specifies an origin-0 index of the first number to extract, and the number of elements required. The index is a single integer, applied to a ravelled form of the input array.

The input and output types supported are:

A0  APL object
B1  Boolean (1 bit, packed 8 per byte)
B8  8-bit binary (unsigned)
I2  halfword binary
I4  fullword binary
E4  1-word floating point
E8  2-word floating point
EX  4-word floating point
C0  An item from a CDR (input only)

Here is a summary of the parameters for this service call. Like all services, the AP must provide a standard CALL-type interface.

1. DN  The two-character service request code.
2. A fullword service completion code.
3. A fullword containing the length of the output buffer.
4. The numeric results.
5. The data to be converted, except that for type A0 it is the CDR or VS APL descriptor of the data.
6. Two two-byte fields, each containing a two-character data type code. The first field determines the format of the 5th parameter, while the second determines the format of the 4th parameter.
7. A fullword containing an origin-0 index into the input data.
8. A fullword containing the count of elements to be converted.
9. For type C0 only, the simple (never G-type) CDR descriptor of the input array.
Error Handling Services

There are three services in this group:

ED  Produce a dump (but continue processing)
ET  Terminate abnormally
EX  Set or clear an ABEND exit

All three services have very simple parameter lists. The rules for exit routines defined by the EX service are more complicated.

ED: Produce a Dump

1. ED  The two-character service request code.
2. A four-character dump identifier.
3. An optional eight-byte Program Status Word associated with the problem.
4. An optional 16 word area containing register values associated with the problem.

ET: Terminate Abnormally

Note: If an EX exit currently exists for the process requesting the ABEND, that exit routine will gain control. You may want to clear the exit using the EX service before issuing ET.

1. ET  The two-character service request code.
2. A fullword containing an abend code number between 1 and 999.

EX: Set or Clear an ABEND Exit

This service specifies the address of an exit routine which will be given control if an ABEND or program check occurs while the process is in control. Any previous exit for the same process is cleared when an exit is set (that is, there is no facility for stacking exits).

The exit routine is not given control on attention signals unless the process is terminated because of repeated unacknowledged signals. The abend exit will be given control even on nonretryable abends for which APL2 gains control. On an MVS system these include operator cancel, timeout, etc. In general, VM does not give APL2 control in nonretryable situations.

1. EX  The two-character service request code.
2. A fullword containing the address of the routine to be given control, or zero to remove the abend exit for this process.

Entry/exit conditions for abend exits

The abend exit is entered using a normal CALL interface, and the following parameter list:

1. A fullword containing the user or operating system abend code.
2. A fullword in which a retry address may optionally be supplied. If it is not, the process will be terminated on return from the exit.
3. A four character field in which a dump code may optionally be supplied to request a dump on exit.
4. On entry to the exit routine, the registers as of the last service call issued by the processor. On exit, the registers that will be passed to the retry routine.
5. An abend type indicator: F (Force off), P (Program check), S (System abend), or U (User abend).
6. For type P only, the hardware Program Status Word (PSW) at the time of the error.
7. For type P only, registers that correspond to the PSW in the 3rd parameter.
File System Services

The file system provided through these services corresponds to that used by AP 121.

**FC** Create an APL File

1. FC The two-character service request code.
2. A fullword service completion code.
3. A fullword library number within which the file is to be created.
4. An 8-character field containing the name of the file to be created.
5. An 8-character field containing an optional password for the library.
6. A fullword containing the maximum size of the file in bytes, or zero.
7. A 2-character field in which the second byte must contain an S or D to indicate a Sequential or Direct file.
8. A fullword containing, for Direct files, the maximum length (in bytes) that any record in the file will ever require.

**FD** Delete an APL File

1. FD The two-character service request code.
2. A fullword service completion code.
3. A fullword library number within which the file is to be deleted.
4. An 8-character field containing the name of the file being deleted.
5. An 8-character field containing an optional password for the library.

**FS** Change the Size of an APL File

1. FS The two-character service request code.
2. A fullword service completion code.
3. A fullword library number within which the file exists.
4. An 8-character field containing the name of the file being changed.
5. An 8-character field containing an optional password for the library.
6. A fullword containing the new maximum size of the file in bytes.

**FA** Open an APL File

1. FA The two-character service request code.
2. A fullword service completion code.
3. A fullword library number within which the file exists.
4. An 8-character field containing the name of the file to open.
5. An 8-character field containing an optional password for the library.
6. A fullword *file token*. This value must be provided on subsequent FR and FW requests for the file, and must be "turned in" on the FZ request that closes the file.
7. A 2-character field in which the first character is R for read-only access or W for read/write access, and the second one contains an S or D to indicate whether the file will be processed sequentially or by direct access.
8. An optional fullword in which the service will return the maximum length (in bytes) that any record in the file can ever use.

9. An optional fullword in which the service will return the number of records that currently exist in the file.

**FZ: Close an APL File**

1. FZ The two-character service request code.
2. A fullword service completion code.
3. (reserved)
4. (reserved)
5. (reserved)
6. A fullword containing the token provided when the file was opened.

**FR: Read an APL File Record**

1. FR The two-character service request code.
2. A fullword service completion code.
3. A fullword containing the length of the area pointed to by the 4th parameter.
4. An area in which the record will be returned, beginning with a four byte length field.
5. A fullword containing the relative record number in the file if the file was opened for direct processing. For sequential processing this value is returned by the system.
6. A fullword containing the token provided when the file was opened.

**FW: Write an APL File Record**

1. FW The two-character service request code.
2. A fullword service completion code.
3. (reserved)
4. An area which contains the record to be written, beginning with a four byte length field.
5. A fullword containing the relative record number in the file if the file was opened for direct processing. For sequential processing this value is returned by the system.
6. A fullword containing the token provided when the file was opened.
Message Services

These services give processors access to the same message facilities used by the APL2 product. Messages may be displayed, queued, or returned to the caller. The current national language table is used, substitution fields are supported, and a message ID is optionally supplied.

The two message services are:
- MC: Check for Message Existence
- MF: Format a Message

MC: Check for Message Existence

The message number "exists" if it can be found in either the standard English table provided as a part of the product or the current national language definition as selected by $NL^T$. Note that this service provides a return code for an unknown message number, while the MF service abends in that case.

1. MC The two-character service request code.
2. A fullword service completion code.
3. A fullword message number.

MF: Format a Message

This service formats a message, then either displays it, queues it, or returns it to the caller. The service depends on a message number as defined in "APL2 Messages and Codes." In the future it will also be possible to define new message numbers in message files selected by $NL^T$.

A one-character code indicates what should be done with the message:
- D Display the message as a part of the APL session.
- Q Queue the message for a subsequent )MORE request.
- R Return the formatted message to the caller.

In each of these cases the message will begin with a message ID if DEBUG(32) is in effect.

This service has two different parameter structures, depending on the action code. For code D or Q the parameters are:

1. MF The two-character service request code.
2. A fullword service completion code.
3. A fullword message number.
4. A single character D or Q.
5. An optional string to be substituted into the message. Message substitution fields are numbered in the message models.
6. A fullword containing the length of the preceding string.

Additional pairs of parameters like 5 and 6 may be provided to define additional substitution strings.

For code R the parameters are:

1. MF The two-character service request code.
2. A fullword service completion code.
3. A fullword message number.
4. A single character R.
5. The output area for the message.
6. A fullword containing the length of the output area. On return this will contain the length of the message.

7. An optional string to be substituted into the message.

8. A fullword containing the length of the preceding string.

Additional pairs of parameters like 7 and 8 may be provided to define additional substitution strings.
Process Services

Process services use one-word blocks called event control blocks (ECBs) to synchronize the operations of two processes. The Shared Variable service and Terminal services also uses ECBs. The internal format and content of an ECB is system dependent, but may be partially controlled by the POSTing process.

There is no return code from any of the processor services. Information about the success of the operation is often available in an ECB. Invalid parameters cause an ABEND of the processor.

PW: Wait for an Event

If multiple ECBs are specified, control may be returned when any one of them has been posted.

1. PW The two-character service request code.
2. A fullword in which a pointer to a posted ECB will be returned.
3. A fullword ECB which is to be posted asynchronously by another task.
4. An optional additional ECB or ECBs. (This is a variable length parameter list.)

PP: Post an ECB

Send a signal to another task in the same address space or virtual machine. This signal will terminate an operating system WAIT or a PW service that has suspended any task on that ECB. It will also set a post bit in the ECB so that a later WAIT or PW will complete immediately.

1. PP The two-character service request code.
2. A fullword ECB which is to be posted.
3. A fullword containing a nonnegative binary number which will be placed in the low order halfword of the ECB.

PT: Start a Timer

This request sets an "alarm clock" which will send a signal after a specified amount of "wall clock" time has elapsed. A timer that has not expired is cancelled by a subsequent timer request from the same process, or by the process's termination.

Control returns immediately, although normally the ECB will not yet have been posted. Use the PW service to wait for the timer signal.

1. PT The two-character service request code.
2. A fullword ECB which will be posted when (or soon after) the time interval has elapsed.
3. A fullword containing the length of time, in milliseconds.
Shared Variable Service

This service provides communication and data transfer between auxiliary processors and the SVP. The parameter list itself is very simple, but the second parameter is a more complex parameter block. One of three different parameter blocks must be provided there, depending on the type of request being made. In all cases the first halfword of the parameter block identifies the request, and hence the format of the remainder of the block. The three parameter blocks are associated with three classes of requests.

SC: Shared Variable Service

1. SC The two-character service request code.
2. A processor control vector (PCV) or share control vector (SCV) or SVP data format block (SDF).

PCV: Processor Requests

Processor requests are related to the state of the auxiliary processor itself, without reference to particular shared variables. The two processor requests are CSVON (signon) and CSVOFF (signoff).

The PCV contains the following fields:

PCVREQ CSVON or CSVOFF.
PCVID Processor identification.
PCVECB Pointer to an event control block.
PCVSPQ Space quota.
PCVSHVQ Shared variable quota.
PCVRC Return code.
PCVOFFER Set if one or more incoming offers exist at the time of signon.

SDF: Data Format Request

This request permits data compatibility with other APL systems. It would be used to request data in a VS APL format, or to return to the default APL2 format. The request applies to individual shared variables.

The SDF contains the following fields:

SCVREQ CSVDFORM
SDFID Processor ID.
SDFPSX The value returned in SCVPSX.
SDFVERS Processor version, always 2.
SDFDFORM Data format to be used. 1 (VS APL) or 2 (APL2).
SDFRC Return code.
SCV: Shared Variable Requests
These are the "workhorse" requests: they handle all shared variable connection, status, and data transfer.

In all of the share requests, the SVP uses a value called the "pershare index" to associate the request with a specific shared variable. When a new variable is being offered, the SVP returns an internally generated pershare index to the caller. It also returns a pershare index for each variable reported in response to CSVSCAN or CSVQUERY. For all other share requests the caller must provide a pershare index previously returned by the SVP.

The SVC requests are:

CSVSCAN Scan for an offer
CSVSHARE Offer a variable, match an incoming offer, or obtain information about a share.
CSVSEEAC See (inspect) access information.
CSVSETAC Set the access control vector.
CSVREF Reference a shared variable.
CSVSPEC Specify a shared variable.
CSVCOpy Copy a value without signalling a reference, and optionally place a hold on the variable.
CSVREL Release a previous hold on the variable.
CSVRET Retract the share offer for a variable.
CSVQUERY Obtain a list of processors or variables which match a specified degree of coupling.
CSVSTATE Obtain information about the state of a list of variables.

The SCV contains the following fields, but only a subset of them is used by each request:

SCVREQ One of the requests listed above.
SCVRC Return code.
SCVPART Partner identification.
SCVID Processor identification.
SCVOSN Offer sequence number.
SCVPSX The pershare index.
SCVECB Pointer to an event control block.
SCVVLEN Shared variable value length, or length of the area pointed to by SCVVALUE.
SCVVALUE Pointer to the shared variable value. For CSVSTATE or CSVQUERY this is a buffer where a list of entries will be returned.
SCVACV Access control vector component.
SCVNAMES On if any name is acceptable.
SCVHOLD The variable will remain under the control of the requestor.
SCVFISPC Ignore any unreferenced value set by the partner.
SCVFOR1 Offered by this processor.
SCVSHR Fully shared.
SCVFOR2 Offered to this processor.
SCVFLGS2 The partner protocol (1=VSAPL, 2=APL2).
SCVLLEN Name length.
SCVNAME Pointer to Shared Variable Name field.
Terminal Services

Two terminal services are defined, TA which allocates the session terminal, and TZ which releases it. Actual terminal I/O must be accomplished with non-APL services such as GDDM or specific operating system interfaces. APL has no way of verifying that auxiliary processors bracket their terminal I/O with proper TA and TZ calls, but if they do not the results may be visually unpredictable, and asynchronous interrupts may not be handled properly.

APL2 will delete any terminal attention exits of its own before giving terminal control to the process. The processor must delete any attention exits it establishes before returning terminal control to APL2.

**TA: Allocate the Terminal**

This is a request for exclusive use of the terminal. The request returns immediately, whether or not the terminal can be given to the requestor at the moment.

On return, the requesting program should wait for a signal indicating that the request has been granted. The PW service can be used for this purpose. A terminal state code will be provided when the signal is sent. It will contain one of:

- **D** Data displayed on the screen has been changed since the processor last controlled it, but field definitions are still valid.
- **F** Field definitions have been changed since the processor last controlled the screen.
- **N** No screen changes have occurred since the processor last controlled the screen, or this is not a full screen terminal, or the processor has never previously controlled the terminal.

The requesting process will retain control of the terminal until it explicitly relinquishes that control with a TZ request. It may receive a signal indicating that some other process is requesting control of the terminal.

1. **TA** The two-character service request code.
2. A one-character field, indicating the state of the terminal. This value will be supplied when terminal control has been granted.
3. A fullword ECB which will be posted when the requestor is given control of the terminal.
4. A fullword ECB which will be posted if some other process requests control of the terminal while this process is holding it.

**TZ: Release the Terminal**

1. **TZ** The two-character service request code.
2. A one-character field indicating what changes have been made to the terminal while it was held. The values are as defined for TA above.
Virtual Storage Services

Storage obtained by these services is always initialized to binary zero. There are no return codes from the services, except that a returned storage address of zero means the requested storage was not available. If invalid parameters are provided, an ABEND will be issued.

VP: Get Process Storage

Storage obtained through this service will be implicitly freed when the process terminates.

1. VP The two-character service request code.
2. A fullword containing the number of bytes of storage needed.
3. A fullword in which the address of the storage is returned.

VG: Get Global Storage

Storage obtained through this service will be retained until APL2 session termination, even if the process terminates earlier.

1. VG The two-character service request code.
2. A fullword containing the number of bytes of storage needed.
3. A fullword in which the address of the storage is returned.

VF: Free Global Storage

1. VF The two-character service request code.
2. A fullword containing the number of bytes of storage to free.
3. A fullword containing the address of the storage to be freed.

VQ: Free Process Storage

1. VQ The two-character service request code.
2. A fullword containing the number of bytes of storage to free.
3. A fullword containing the address of the storage to be freed.

VV: Get Variable Length Process Storage

This request is identical to VP except that a smaller amount of storage will be accepted if the amount requested is not available.

1. VV The two-character service request code.
2. A fullword containing the maximum number of bytes of storage wanted. On return it will contain the number of bytes actually obtained.
3. A fullword in which the address of the storage is returned.
Example of an Auxiliary Processor

The code shown here is a usable auxiliary processor (though it may still have some bugs in it). The processor provides a simple file system, with one arbitrarily complex APL2 data array per file. To use the system, share one variable with AP 421. The variable name is a one to eight character file name. Multiple concurrently shared variables are not supported.

To write a file:
- Assign a two item nested vector to the shared variable:
  1. A library number expressed as a one element vector.
  2. The data to be stored, of arbitrary structure.
- The AP returns a one element vector numeric return code, using the return codes defined for AP 121.
- Storing an array replaces any previous data in the file.

To read a file:
- Assign a single element numeric vector to the shared variable, representing a library number.
- If the operation is successful, the AP returns a two item nested vector, exactly as as it was provided when the file was written. (The first item is the original library number.)
- If the operation fails, the AP returns a one element vector numeric return code, using the return codes defined for AP 121.
Initialization of the AP

FILESAMP CSECT
    SAVE (14,12), --FILESAMP-CSEC-8SYSDATE
    LR RBASE,R15
    USING FILEAMP,RBASE
    LR RPARM,R1
    USING PARMs,RPARM
    LA QOPS,1 SET UP TO FORCE 0C6
    * NOTE: CHANGE ABOVE TO 0 TO FORCE LOOPS INSTEAD
    *** GET A WORK AREA FOR OURSELVES
    L R15,PARM4 SET UP STORAGE REQUEST
    LA R8,WORKLEN - LENGTH REQUIRED
    ST R8,0(R15)
    L R15,PARM2 FIND ADDRESS OF SERVICE ROUTINE
    L R15,0(R15) - IT IS AN INDIRECT POINTER
    LR RSERV,R15 (AND SAVE FOR THE FUTURE)
    LA R1,PARM3 USE AS 'VP' PARM LIST
    CALL (15) GET WORK STORAGE
    L - R15,PARM5 FIND ADDRESS OF STORAGE
    L RWORK,8(R15) USING WORK, RWORK
    * NOTE: IF THERE IS NO STORAGE, WE WILL BLOW UP SHORTLY,
    WHICH IS AS GOOD A WAY AS ANY FOR US TO COMPLAIN.

Set Up to Use the SVP

    LA R8,VNAME
    ST R8,SCVNAME WHERE NAME SHOULD GO
    LA R8,SECB SHARED VARIABLE ECB
    ST R8,SCVECB
    *** SIGN ON TO THE SVP
    LA R8,421
    ST R8,PCVID OUR ID (2ND WORD ALREADY 0)
    ST R8,SCVID - WILL WANT IT IN SVC, TOO
    LA R8,PECB PROCESSOR ECB
    ST R8,PCVECB
    SR R8,R0
    BCTR R8,0 WE HANDLE VARIABLES OF ANY SIZE
    SRL R8,8 50 SAY 2**24 - 1
    ST R8,PCVSPQ
    LA R8,1 BUT ONLY ONE VARIABLE AT A TIME
    STH R8,PCVSHVQ
    LA R8,CSVON
    STH R8,PCVREO ASK FOR SIGNON
    LR R15,RSERV
    CALL (15),((E'SC', PCV),VL, MF=(E,WORKA)
    LH R15,PCVRC
    LTR R15,R15 CHECK RETURN CODE
    BIHZ SHUT9 GET OUT IF CAN'T SIGN ON
Wait for an Offer, and Terminate when Required

Phait  DS  0H
LR  R15,RSERV
CALL (15),('C'PW',DUMMY,PECB),VL,MF=(E,WORKA)
SR  R8,R8
ST  R8,PECB  CLEAR ECB FOR THE NEXT POST
ST  R8,SCVOSN  LOOK FOR ANY OFFER
MVI  SCVFLGS1,SCVNAME  ANYTHING IS OK
LA  R8,L'YNAME
STC  R8,SCVLEN  MAX NAME LENGTH
LA  R1,CSVSCAN
BAL  RBACK,CALLSVP  SCAN FOR AN OFFER
BZ  MATCH  - TRY TO MATCH IF OFFER
BNO  Phait  - GO WAIT UNLESS SHUTTING DOWN

***  TERMINATE THE AUXILIARY PROCESSOR  ***

Shut  DS  0H
LA  R8,CSVOFF
STH  R8,PCVREQ  ASK FOR SIGNOFF
LR  R15,RSERV
CALL (15),('C'SC',PCV),VL,MF=(E,WORKA)

Shut  DS  0H
*  NOTE: OUR WORKAREA IS FREED AUTOMATICALLY
RETURN (14,12)

Match an Incoming Offer

Match  DS  0H
*  NOTE: SCV ALREADY CONTAINS INFO ABOUT THE OFFER
*  USE NAME AS A FILE NAME
LA  R14,VNAME  HERE IS WHERE THE NAME IS
LA  R15,C' '  -(PAD WITH BLANKS)
SLL  R15,24
IC  R15,SCVLEN  -GET ITS LENGTH
LA  R8,FNAME
LA  R1,L'FNAME  MOVE IT INTO FILE FIELD
MVCL  R8,R14
SR  R8,R8
ST  R8,SCVLEN  WE DON'T HAVE AN INITIAL VALUE
MVI  SCVACV,B'0110'  CONTROL HIS SET, MY USE
LA  R1,CSVSHARE
BAL  RBACK,CALLSVP  MATCH AN OFFER
BO  Shut  - GET OUT IF SHUTTING DOWN
BNZ  Phait  - IGNORE IF CAN'T MATCH
Get and Analyze a Request from the User

HANDLE DS 0H
   LA R1,256
   MINIMUM BUFFER SIZE

HAND2 DS 0H
   BAL RBACK,GETBUFF
   BZ EMSG
   GO EXPLAIN IF NO SPACE
   USING BUFFER,RBUFF
   LA R1,CSVREF
   BAL RBACK,CALLSVP
   BZ HAND4
   REFERENCE THE VARIABLE VALUE
   BO SHUT
   - PROCESS IF WE HAVE A VALUE
   BM WARENT
   - GET OUT IF NO SVP
   L R1,SCVLEN
   - GO WAIT IF INTERLOCK
   C R1,BUFLEN
   B HAND2
   ELSE PROBABLY BUFFER PROBLEM
   B *(OOPS)
   - SO IT IS, TRY AGAIN
   - OUR PROBLEM IF NOT

HAND4 DS 0H
   CLI ,CDRRT,RTG
   BNE READ
   THEN GO HANDLE AS INPUT
   B WRITE
   ELSE GO HANDLE AS OUTPUT

Wait for Action on the Current Share

WAIT DS 0H
   TH SCVFLGS1,SCVFSHR
   BZ RETR
   RETRACT IF PARTNER DID
   LR R15,RSERV
   CALL (15),(-C'PH',DUMMY,SECB,PECB),VL,HY=(E,WORKA)
   SR R8,RO
   ST R8,SECB
   B HANDLE
   AND SEE WHAT WE HAVE NOW

***
   RETRACT THE SHARE, SINCE PARTNER ALREADY HAS
   *

RETR DS 0H
   LA R1,CSVRET
   BAL RBACK,CALLSVP
   RETRACT THE VARIABLE
   BO SHUT
   - GET OUT IF SHUTTING DOWN
   BZ WAIIT
   - GO TO PRIMARY WAIT IF OK
   B *(OOPS)
   ELSE WE'VE GOT A PROBLEM HERE
Write a File

```
WRITE
DS 0H
LA R8,2
C R8,CDRXRH0 CHECK FOR A TWO ITEMS
BNE EMSG NOT SLAP HIS HAND
LA R8,1
CH R8,CDRRANK+LIBDESC FIRST SHOULD BE A VECTOR
BNE EMSG
C R8,CDRXRH0+LIBDESC WITH EXACTLY ONE ELEMENT
BNE EMSG
BAL RBACK,GETNUM GET LIBRARY NUMBER
BNZ EMSG IF NOT AN INTEGER, BLAME THE USER

DELETE, RECREATE, OPEN, WRITE, AND CLOSE FILE

LR R15,RSERV
CALL (15), ("C'FD'",RC, DELETE THE FILE IF IT EXISTS +
LIBNO,FNAME,PASS), -IDENTIFY THE FILE +
VL,MF=(E,WORKA)

NOTE: RETURN CODE IGNORED. MAY BE NOT FOUND.

LR R15,RSERV
CALL (15), ("C'FC'",RC, RECREATE THE FILE +
LIBNO,FNAME,PASS, -IDENTIFY THE FILE +
=F"0",=C'WS' ), -UNLIMITED SIZE, SEQUENTIAL +
VL,MF=(E,WORKA)

LTR R15,R15
BNZ EMSG BE SURE THAT WORKED

LR R15,RSERV
CALL (15), ("C'FA'",RC, OPEN THE FILE +
LIBNO,FNAME,PASS, -IDENTIFY THE FILE +
TOKEN,=C'WS' ), -ASK FOR SEQUENTIAL WRITE +
VL,MF=(E,WORKA)

LTR R15,R15
BNZ EMSG BE SURE THAT WORKED

L R8,SCVLEN
ST R8,RECLLEN PASS ALONG ARRAY SIZE

LR R15,RSERV
CALL (15), ("C'FW'",RC, WRITE FIRST RECORD +
,(RBUFF), -ADDRESS OF RECORD +
DUMMY,TOKEN), -USE FILE JUST OPENED +
VL,MF=(E,WORKA)

LR RTemp,R15 SAVE RETURN CODE

LR R15,RSERV
CALL (15), ("C'FZ'",RC, CLOSE THE FILE +
,...TOKEN), -IDENTIFY THE FILE +
VL,MF=(E,WORKA)

LR R15,RTemp RETURN CODE (MAY BE ZERO)
B EMSG GO SAY WHAT HAPPENED

PASS DC CL8', WE NEVER SUPPLY PASSWORD
```
Read a File

```
READ DS 0H
LA R0,1
C R0,COPYRHO CHECK FOR A SINGLE ITEM
BNE EMSG NO? SLAP HIS HAND
* NOTE: GETNUM WILL CATCH ANY CHARACTER DATA
* WE LET SINGLE NUMERIC ITEMS OF ANY RANK THROUGH
BAL RBACK,GETNUM GET LIBRARY NUMBER
BNZ EMSG IF IT DIDN'T WORK, BLAME THE USER
LR R15,RSERV
CALL (15),(<'FA' ,RC OPEN THE FILE
LIBNO,FNAME,PASS, -IDENTIFY THE FILE
TOKEN,<'RS'), -ASK FOR SEQUENTIAL READ
VL,MF=(E,WORKA)
LTR R15,R15
BNZ EMSG
READ2 DS 0H
LR R15,RSERV
CALL (15),(<'FR',RC READ FIRST RECORD
BUFLEN,(RBUFF), -LENGTH/ADDRESS TO READ TO
DUMMY,TOKEN), -USE FILE JUST OPENED
VL,MF=(E,WORKA)
LR RTEMP,R15 SAVE RETURN CODE
LA R0,ERLEN
CR R0,R15 IF NOT RECORD LENGTH ERROR
BNE READ5 THEN GO ON AND CLOSE
L R1,BUFLEN
AR R1,R1 ELSE DOUBLE THE ANTE
BAL RBACK,GETBUF
BNZ READ2 TRY AGAIN IF THAT WORKED
B EMSG ELSE GO EXPLAIN
READ5 DS 0H
LR R15,RSERV CLOSE THE FILE
CALL (15),(<'FZ',RC,...,TOKEN),VL,MF=(E,WORKA)
LTR R15,RTEMP
BNZ EMSG
* RETURN RESULT IN SHARED VARIABLE
L R0,RECLEN
ST R0,SCVLEN LENGTH OF THE CAR
LA R0,CDR
ST R0,SCVVALUE ADDRESS OF THE CDR
HVI SCVFICS1,0 DON'T IGNORE ANOTHER PARTNER SPEC
LA R1,CVSPEC
BAL RBACK,CALLSVVP SPECIFY THE VARIABLE VALUE
BZ WAIT - WAIT FOR NEXT REQUEST IF OK
BH READ8 - EXPLAIN IF INTERLOCK OR SM FULL
B0 SHUT - GET OUT IF NO SVP
CH R15,=#(CSEVOS) IF PARTNER HAS SET VARIABLE AGAIN
BE WAIT THEN IGNORE THIS AND DO THAT
CH R15,=#(CSEVTL) IF VALUE NOT TOO LARGE FOR SM
BNE * (OOPS) THEN WE GOOFED
READ8 DS 0H
LA R15,ERSPC STRANGE, BUT AIP121 TREATS SM FULL
B EMSG AS NO STORAGE FOR I/O BUFFER
```
Tell the User What Happened

This routine attempts to send a message to the terminal if the return code (currently in R15) is nonzero. It also assigns the return code to the shared variable in all cases.

EMSG  DS  0H
  MVC WORKCDR,RCCDR  SET UP RESULT CDR
  ST R15,WORKCDR+RCDATA-RCCDR  FILLING IN RETURN CODE
  LTR R15,R15  IF SUCCESSFUL COMPLETION
  BZ EMSG5  THEN SKIP THE MESSAGE
  CVD R15,DWORD
  MVC STRING,EMASK
  EDWK STRING,DWORD+6  CONVERT RETCODE TO CHARACTER FORM
  LR RTEMP,R1
  LA R0,STRING+L'STRING
  SR R0,RTEMP  LENGTH OF RESULT
  ST R0,MLEN
  LR R15,RSERV
  CALL (15),('MF', CALL MESSAGE SERVICE
               '22', 'C'D', PROCESSOR ___ ERROR ___
               +
               PROCM,PROCLEN,
               (RTEMP),MLEN),
               +
               VL,MF=(E,WORKA)

* NOTE: IF IT WORKED, FINE. IF NOT, QUE SERA, SERA.

EMSG5  DS  0H
  LA R0,WORKCDR
  ST R0,SCVVALUE
  LA R0,WORKCDR
  ST R0,SCVLEN
  LA R1,CSVSPEC
  BAL RBACK,CALLSVP
  BZ WAIT  - WAIT FOR NEXT REQUEST IF OK
  BH WAIT  - GIVE UP IF INTERLOCK OR SM FULL
  BO SHUT  - GET OUT IF NO SVP
  CH R15,Y(CSVVOS)  IF PARTNER HAS SET VARIABLE AGAIN
  BE WAIT  THEN IGNORE THIS AND DO THAT
  B *(OOPS)  OTHERWISE WE GOOFED

EMASK  DC X'48292125'
PROCHO DC C'421'
PROCLEN DC A('PROCHO')
RCCDR DC AL1(CDRID),AL3(RCDATA-RCCDR)  HEADER
       DC F'1'
       DC AL1(RTI,RL4)  TYPE AND LENGTH
       DC H'1'
       DC F'1'
       DC F'1'
RCDATA DC F'8'
RCLEN EQU **-RCCDR
Subroutine to Call the SVP for a Share Service

R1 must contain an SVP service code, and the SCV must be set up. This routine returns with the machine condition code set to "overflow" if the SVP is not active. In all other cases, it is set to the sign of the SVP return code. The SVP returns zero on success, negative numbers on temporary problems (such as shared variable interlock), and positive numbers on permanent problems.

CALL SVP DS 0H
STH R1, SCVREQ INDICATE DESIRED REQUEST
LR R15, RSERV
CALL (15), (-E'SC', SCV), VL,MF=(E, WORKA)
LH R15, SCVRC
LTR R15, R15 CHECK RETURN CODE
BNPR RBACK GET OUT IF CAN'T BE UNAVAILABLE
LA R8, CSVENA
CR R15, R8 CHECK FOR UNAVAILABLE
BHR RBACK LEAVE CC=P IF NOT UNAVAILABLE
TM *1,X'FF' A TEST THAT CAN'T FAIL
SR RBACK RETURN WITH CC=0

Subroutine to Extract a Library Number from a CDR

The LIBNO field is set to the first data item in the CDR. This routine returns with the machine condition code set to zero if the number was gotten successfully. Failure would normally occur because the first item was in turn nested, or contained character data, or contained a number which could not be converted to an integer.

GETNUM DS 0H
L RTEMP, CORDLEN DESCRIPTOR LENGTH
LA RTEMP, CDR(RTEMP) POINT AT DATA, CLEAR HIGH BIT
LA RTEMP2, CORDRES ASSUME SIMPLE CDR
CLI CDRRT, RTG IF IT IS NOT NESTED
BNE GETN2 THEN USE THAT ITEM
LH R1, CDRRANK ELSE STEP PAST THAT ONE
SLL R1, 2 (ALLOWING FOR SHAPE WORDS)
LA RTEMP2, CDRRHO(R1)

GETN2 DS 0H
LR R15, RSERV
CALL (15), (-E'ON', RC, GET THE NUMBER
=A(L'LIBNO), LIBNO, OUTPUT AREA LENGTH AND ADDRESS
(RTEMP), -INPUT DATA ADDRESS
=CCBO14', -CONVERT CDR TO FULLWORD INTEGER
=F'0',F'1', -ELEMENT 0 FOR 1 ELEMENT
(RTEMP2)), -DESCRIPTOR FOR COMPO N NUMBER
VL,MF=(E, WORKA)
LTR R15, R15 INDICATE WHAT HAPPENED
BZR RBACK AND RETURN IF OK
CH R15, =W'H13' DATA COULDN'T BE CONVERTED?
BNE *(DOPS) ANYTHING ELSE IS OUR PROBLEM
LA R15, ERSYN NOT CONVERTED IS SYNTAX ERROR
LTR R15, R15 INDICATE WHAT HAPPENED
BR RBACK AND RETURN
Subroutine to Get a Buffer

This routine is called with R1 containing the length required. It checks any existing buffer to see if it is big enough. If not, it frees the old one (if any) and gets a new one. It then sets up all the fields that expect to have buffer addresses or lengths. This routine returns with the machine condition code set to zero if no storage could be gotten. R15 is set to the AP 121 error code for "no space" in that case.

```
GETBUFF DS 0H
    L RBUFF,BUFPTR  SAY WHERE BUFFER IS
    C R1,BUFLEN
    BNE GETB8  SKIP MOST IF CURRENT BUFF IS OK
    LA RTEMP,R1  ELSE REMEMBER SIZE NEEDED
    L R0,BUFLEN
    LTR R0,R0  IF NO BUFFER AT PRESENT
    BZ GETB2  THEN GO GET ONE
    LR R15,RSERV
    CALL (15),(-C'Q',BUFLEN,BUFPTR),  FREE PRIOR BUFFER
    V1,MF=(E,WORKA)  +

GETB2 DS 0H
    ST RTEMP,BUFLEN
    LR R15,RSERV
    CALL (15),(-C'VP',BUFLEN,BUFPTR),  GET A NEW BUFFER
    V1,MF=(E,WORKA)  +
    L RBUFF,BUFPTR  SAY WHERE BUFFER IS
    LTR RBUFF,RFUFF  IF WE MANAGED TO GET ONE
    BNZ GETB8  THEN GO AHEAD
    LA R15,RSERV  ELSE SAY NO SPACE
    BR RBACK  AND RETURN (CC=0 FROM LTR)

GETB8 DS 0H
    LA R0,CDR
    ST R0,SCVVALUE  SAY WHERE SVP DATA GOES
    LR R1,RFUFF
    A R1,BUFLEN  - END OF BUFFER
    SR R1,R8  - LENGTH OF CDR AREA
    ST R1,SCVLEN  SAY HOW BIG SHARED VARIABLE CAN BE
    BR RBACK  AND RETURN (CC=0 FROM SR)
```

Data Declarations

Hiding back here at the end are all the extra things you have to tell the assembler to convince it to produce a working program. Incidentally, the fact that they are at the end probably means that the H assembler is required for compilation.

First the definitions which do not use storage.
REJ E'JU e Rl EQU 1
REU EQU 2
RTEnp2 EQU 3
OOPS EQU 4
RBUFF EQU 5
RST REV EQU 6
RPARM EQU 7
RWORK EQU 8
RBACK EQU 9
RBASE EQU 10
R13 EQU 11
R14 EQU 12
R15 EQU 13
RLEN EQU 14
ERSYN EQU 15
ERSPC EQU 16
AP2CSVPE EQU 17

FILE SERVICE RETURN CODES
ERLEN EQU 18
ERSYN EQU 19
ERSPC EQU 20

AP2CSVPE , SVP SERVICE CODE DEFINITIONS

The remaining definitions describe storage based on some pointer. Note, incidentally, that this AP is completely reentrant.

BUFFER DSECT , FOR FILE I/O AND SHARED VARIABLE REF/SPEC
RECLEN DS F LENGTH WORD FOR FILE SYSTEM
AP2CDR TYPE=CSECT,DOC=NO SHARED VARIABLE VALUES
ORG CDRED
DS F LENGTH OF G ITEM
CDRDSECS EQU * START OF SECOND DESCRIPTOR
LIDBESC EQU *-CDRES OFFSET TO LIBRARY NUMBER ITEM

PARMS DSECT , PARMS PASSED TO EACH TASK
PARM1 DS A (UNUSED)
PARM2 DS A INDIRECT PTR TO SERVICE ROUTINE
PARM3 DS A START OF 'VP' PARM LIST
PARM4 DS A - LENGTH FOR VP
PARM5 DS A - ADDRESS FROM VP

WORK DSECT , LOCAL WORK AREA
DHORD DS D DECIMAL ARITHMETIC WORK AREA
STRING DS CL(L'CMDASK) DECIMAL DISPLAY WORK AREA
MLEN DS F LENGTH OF DATA IN STRING
BUFFPR DS A POINTER TO BUFFER
BUFFLEN DS F LENGTH OF BUFFER
PECB DS F PROCESSOR EVENT CONTROL BLOCK
SECB DS F SHARED VARIABLE EVENT CONTROL BLOCK
VNAME DS CL8 SHARED VARIABLE NAME
FNAME DS CL8 FILE NAME
LIBNO DS F LIBRARY NUMBER
TOKEN DS F FILE SERVICES TOKEN
WORKA DS 9A SERVICE CALL PARM LIST AREA
DUMMY DS F UNUSED FIELD REQUIRED BY SOME CALLS
RC DS F RETURN CODE PARAMETER

AP2PCV TYPE=DS PARM BLOCK FOR SVP SIGNON/SIGNOFF
AP2SCV TYPE=DS PARM BLOCK FOR SVP SHARE SERVICES
HORKDRE DS CL(RLENI) RETURN CODE DCR
HORKLEI EQU *-WORK LENGTH OF WORK AREA

Example of an Auxiliary Processor 22