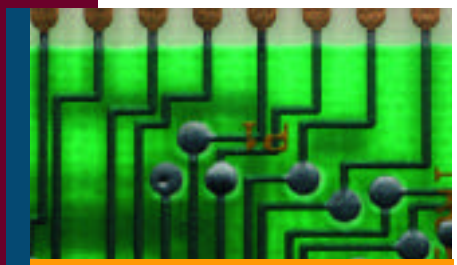

MVS Dynamic Allocation: Part II

BY MICHAEL H. CARROLL

This concluding article examines additional dynamic allocation functions, explains how to handle errors, and presents some real-world examples of how to use/exploit these facilities.

IN Part I (January 1998) I examined the concepts of dynamic allocation and showed the basic methods of allocating and deallocating a dataset. To help you recall the fundamentals, Figure 1 shows sample code to allocate and deallocate a PDS member. If this doesn't make sense, read Part I again. This concluding article examines the other dynamic allocation functions, explains how to handle errors, and presents some real-world examples of how to use/exploit these facilities.

It's interesting to note that if an existing PDS with an non-existent member name is allocated, then the process will succeed. This is the same as for JCL. If the file is subsequently opened for output, all is well, but an attempt to open for input will cause the OPEN to fail. Therefore, always check the success or otherwise of OPEN processing — this is just good programming practice anyway!

After the DYNALLOC function is complete, the dataset should be allocated (or deallocated). After allocation the DD name can be OPENed as normal for processing. However, what happens if an error occurs during the allocation/deallocation process?

ERROR HANDLING

After the invocation of DYNALLOC the result of the SVC 99 call is returned in Register 15. The possible values of Register 15 are outlined in Figure 2. The most basic check that should be performed is to ensure Register 15 contains zeroes. This check is shown in Figure 1 after the DYNALLOC calls. If a non-zero return code is present in Register 15, then the S99ERROR field (if using IEFZB4D0 copybook) or RBERROR field in the request block (if using a scheme like Figure 1) will contain additional information outlining the specific reason for failure.

There are many pages of information on these error codes that are detailed in the IBM manuals (see the references at end of this article). The errors are also listed in the ISPF help facility in Appendix A, which is a quick and useful way to interpret the error information. In most cases, the error may be obvious — dataset does not exist, etc. The categories or classes of error codes are presented in Figure 3.

Quite often, unless you code your program to specifically recognize certain errors, the easiest way to handle this is to display an appropriate message and either terminate or continue depending on what you are doing. This is one of the most powerful aspects of dynamic allocation. If there is a failure, then the program has complete control over whether to continue or not. If using job step allocation, a JCL error may have resulted and the whole job would terminate.

There are a number of ways to handle errors generated by dynamic allocation:

- Place the return code, error code, and information code in a message and either display this on a screen (in an interactive session, for example) or perhaps in the joblog or SYSOUT of a batch job.

- IBM supplies IEFDB476, a program that can interpret the error codes returned by DYNALLOC and generate appropriate messages. The messages can be routed to the TSO user/batch job directly or returned to your program for further handling. This can be linked to after an erroneous DYNALLOC invocation. The Request Block Extension (RBX) is

Figure 1: Sample Code to Allocate and Deallocate a PDS Member

```
*
*           ALLOCATE DSN(MEMBER)
*
            MVI    RBVERB,X'01'             SET UP ALLOCATE
            MVC    RBTEXTA,=A(TEXTLST1)     POINT TO TEXT UNIT LIST
            MVC    TEXTU2+6(44),DSNAME      MOVE IN PDS NAME
            MVC    TEXTU3+6(8),MEMBER       MOVE IN MEMBER NAME
            LA     R1,RBA
            DYNALLOC
            LTR    R15,R15
            BNZ    DYNAERR                  NON-ZERO RESULT - ERROR

            ...
*
*           DEALLOCATE DSN(MEMBER)
*
            MVI    RBVERB,X'02'             SET UP DEALLOCATE
            MVC    RBTEXTA,=A(TEXTLST2)     POINT TO TEXT UNIT LIST
            LA     R1,RBA
            DYNALLOC
            LTR    R15,R15
            BNZ    DYNAERR                  NON-ZERO RESULT - ERROR
            ...
*
*           DYNAMIC ALLOCATION ERROR ROUTINE
*
DYNAERR     EQU    *
            MVC    DYNAMSG+10(5),=C'ALLOC'       SET ERR MSG TO 'ALLOC'
            CLI    RBVERB,X'01'                  WERE WE ALLOCATING ?
            BE     DYNCONV                       YES, LEAVE MESSAGE INTACT
            MVC    DYNAMSG+10(5),=C'FREE '       NO, CHANGE TO 'FREE '
DYNCONV     EQU    *
            CVD    R15,DWORD                     CONVERT RETURN CODE
            UNPK   DYNAMSG+27(2),DWORD           INTO ERROR MESSAGE
            OI     DYNAMSG+28,X'F0'              ENSURE PRINTABLE
            PACK   DYNAMSG+39(1),RBERROR(1)      CONVERT
            MVC    DYNAMSG+40(1),RBERROR         DYNALLOC
            PACK   DYNAMSG+41(1),RBERROR+1(1)    ERROR
            MVC    DYNAMSG+42(1),RBERROR+1       CODE
            NC     DYNAMSG+39(4),OFFZONES        TO
            TR     DYNAMSG+39(4),TRTABHX         PRINTABLE HEX
            PACK   DYNAMSG+55(1),RBINFO(1)       CONVERT
            MVC    DYNAMSG+56(1),RBINFO          DYNALLOC
            PACK   DYNAMSG+57(1),RBINFO+1(1)     INFO
            MVC    DYNAMSG+58(1),RBINFO+1        CODE
            NC     DYNAMSG+55(4),OFFZONES        TO
            TR     DYNAMSG+55(4),TRTABHX         PRINTABLE HEX
            B      DISPMSG
            ...

RBA         DC     0F'0',X'80',AL3(RB)
*
RB          DC     AL1(20)                       Length of Request Block
RBVERB      DC     X'01'                         Verb - 01 for Allelic, 02 for Free
RBFLAGS1    DC     AL2(0)                        Flags 1 - initially set to zero
S99NOMNT    EQU    X'2000'                       Do not mount volumes
S99NOMIG    EQU    X'0100'                       Do not recall migrated file
RBERROR     DC     AL2(0)                        Error Code
RBINFO      DC     AL2(0)                        Information Code
RBTEXTA     DC     A(TEXTLST1)                   Address of text unit list
RBEXTNA     DC     A(0)                          Address of RB extension ( zero )
RBFLAGS2    DC     A(0)                          Flags 2 - initially set to zero
*
TEXTLST1    DC     A(TEXTU1),A(TEXTU2),A(TEXTU3),X'80',AL3(TEXTU4)
TEXTLST2    DC     A(TEXTU1),X'80',AL3(TEXTU7)
*
TEXTU1      DC     X'0001',X'0001',X'0008',CL8'SYSUT1'        DD NAME=
TEXTU2      DC     X'0002',X'0001',X'002C',CL44' '            DSN=
TEXTU3      DC     X'0003',X'0001',X'0008',CL8' '             MEMBER NAME
TEXTU4      DC     X'0004',X'0001',X'0001',X'08'              DISP=SHR
*
TEXTU7      DC     X'0007',X'0000'                            FREE
*
*
DYNAMSG     DC     CL60'* NOTE *  ALLOC ERROR. RC= ** ERR CDE= **** INFO CD*
                   E = ****'
TRTABHX     DC     C'0123456789ABCDEF'
OFFZONES    DC     4X'0F'
```

required to use this facility. See the manuals for complete documentation.

- Allow DYNALLOC to send the message to the end user for you. This involves building an additional control block, the RBX, and placing its address in the field RBEXTNA. Up until now, we have always zeroed this field.

Figure 1 shows an example of the first option while the complete sample application in Figure 5 shows the third method being used.

REQUEST BLOCK EXTENSION

To help you diagnose errors that can occur when using dynamic allocation functions, dynamic allocation supplies information reason codes and IBM supplies an extended message processing program (see the section "Error Handling"). To allow DYNALLOC to send messages directly to the end user when an error occurs, you must code the RBX to receive reason codes in addition to those returned in the S99ERROR (or RBERROR) and S99INFO (or RBINFO) request block fields.

The RBX consists of fields in which you provide information about your request and into which the system stores information about the success of the request. The RBX must begin on a full-word boundary. Mapping macro IEFZB4D0 assigns it a DSECT name of S99RBX.

Unless you plan to let DYNALLOC send messages to the joblog or the TSO user directly outside of your program control, there is no burning need to use the RBX. It is probably useful to see interpreted errors to aid problem diagnosis. I tend to use the home-grown method — i.e., display the return code and error codes myself and look up any errors in the manuals. Generally, the RBX address field is set to zeroes in my request blocks. Figure 5 shows the RBX being used to send messages to the user, in this case a batch job.

REQUEST BLOCK FLAGS

There are two separate sets of flag bytes defined in the RB. The first is identified as RBFLAGS1 (or S99FLAG1 if using IEFZB4D0 copybook) and is a two-byte block. RBFLAGS2 (or S99FLAG2) is a four-byte block and can only be used by authorized programs. The latter is for systems programming use only and should always be set to binary zeroes.

Figure 2: DYNALLOC Return Codes in Register 15

Decimal Code	Meaning and Action
0	Successful completion; there will also be an information reason code returned in S99INFO (or RBINFO) if a non-terminating error occurred during request processing. No further action required.
4	An error resulted from the current environment, the unavailability of a system resource, or a system routine failure; the system will also return an error reason code in Request Block Error Code field (S99ERROR or RBERROR).
8	The installation validation routine denied this request.Contact your systems programmer for more information.
12	The parameter list was not valid; the system will also return an error reason code in S99ERROR or RBERROR.

Figure 3: Categories of Dynamic Allocation Error Codes

Class	Error Code (Hex)	Error Code (Decimal)	Description
1	N/A	N/A	IBM Reserved.
2	02xx	512 - 767	Unavailable system resource - environmental error e.g., 020C - Request for exclusive use of a shared dataset cannot be met.
3	03xx	768 - 1023	Invalid parameter list - program error e.g., 0364 - JOBLIB/STEPLIB/JOBCAT/STEPCAT used as a DD name.
4	04xx	1024 - 1279	Environmnetal error e.g., 0410 - Specified DD name unavailable. Change DD name.
5	N/A	N/A	IBM Reserved.
6	N/A	N/A	IBM Reserved.
7	17xx	5888 - 6143	Catalog LOCATE error e.g., 1708 - Dataset could not be found. (very common).
7	47xx	18176 - 18431	DADSM allocate errors.
7	57xx	22272 - 22527	CATALOG errors.
7	67xx	26368 - 26623	OBTAIN errors.

Figure 4: Sample Request Block (RB) and Text Units to Concatenate Two DD names

```
RBA         DC     0F'0',X'80',AL3(RB)
*
RB          DC     AL1(20)                 Length of Request Block
RBVERB      DC     X'03'                   Verb - 03 for Concatenate
RBFLAGS1    DC     AL2(0)                  Flags 1 - initially set to zero
RBERROR     DC     AL2(0)                  Error Code
RBINFO      DC     AL2(0)                  Information Code
RBTEXTA     DC     A(TEXTLST3)             Address of text unit list
RBEXTNA     DC     A(0)                    Address of RB extension ( zero )
RBFLAGS2    DC     A(0)                    Flags 2 - initially set to zero
*
           DS      0F
TEXTLST3 DC        X'80',AL3(TEXTU1)
*
*                  Key      #        Len        DD      Len        DD
TEXTU1    DC     X'0001', X'0002', X'0008', CL8'DD1', X'0008', CL8'DD2'
```

To help you diagnose errors that can occur when using dynamic allocation functions, dynamic allocation supplies information reason codes and IBM supplies an extended message processing program.

The RBFLAGS1 field has a number of bit masks that turn on certain options concerning dynamic allocation. Generally these are not necessary, but Figure 1 shows a couple of flags that may be of use. These are of particular note:

S99NOMNT — Do not mount volumes or consider offline devices. If not set, then a request to allocate a file that is on a tape or offline DASD would cause program to pause waiting for operator intervention.

S99NOMIG — When set, an allocation attempt on a migrated dataset will fail (with error code X'278'). When not set, migrated dataset will be recalled.

OTHER DYNAMIC ALLOCATION FUNCTIONS

While so far this discussion has concentrated on using DYNALLOC to allocate and deallocate DD name-dataset combinations to our program, however, there are also five other dynamic allocation functions that may be less used, but, nevertheless, can be important occasionally:

Verb Code X'03' - Request for Concatenation: Dynamic concatenation logically connects allocated datasets into a concatenated group. You can identify datasets to be concatenated only by their associated DD names. These datasets must not be open; if they are, the request for dynamic concatenation fails.

DD names are specified in the same order as the system concatenates their associated datasets. The name associated with the concatenated group is the DD name that was specified first; the other DD names are no longer associated with any dataset. If a DD name you specify is already associated with a concatenated group, that entire group is included in the new concatenation.

After the request for dynamic concatenation is satisfied, all members of the dynamically concatenated group are assigned the in-use attribute. This will be described later. An example of the request block and text units for concatenation of two DD names is shown in Figure 4.

Verb Code X'04' - Request for Deconcatenation: Dynamic deconcatenation logically disconnects the members of a dynamically concatenated group. You identify the concatenated group to be deconcatenated by specifying the DD name of the group.

When a concatenated group is dynamically deconcatenated, the DD names that were associated with the datasets before they were concatenated are restored unless this would result in duplicate DD names. This situation could arise if a dynamic allocation with the DD name to be restored occurred after a dynamic concatenation. In this case, the deconcatenation request fails. The request for dynamic deconcatenation also fails if the concatenated group is open. Dynamic deconcatenation has no effect on the in-use attributes associated with the members of the group.

Verb Code X'05' - Request to Remove In-Use Attribute: When a dataset is dynamically allocated, the system assigns it the in-use attribute. You can request that the system remove the in-use attribute by deallocating the dataset or by requesting the remove-in-use function. When the system marks a dynamically allocated dataset as "not-in-use," it does not deallocate the resource.

A dataset that has been marked "not-in-use" becomes eligible for use in a subsequent dynamic allocation request. In addition, the system keeps track of dataset use and knows which datasets have not been in use for the longest time.

The in-use attribute can be removed by specifying verb code 02 with text unit key 8 in the SVC 99 parameter list. This function marks the dataset as " not in use." The in-use flag can also be removed by using this function: (verb code = X'05').

Verb Code X'06'- Request for DD name Allocation: By specifying only the associated DD name, DD name allocation allows you to reuse a previously allocated dataset that was marked "not in use." DD name allocation causes the system to assign the in-use attribute to the dataset.

You request dynamic allocation by DD name by specifying verb code 06 and putting the DD name to be allocated in the DYNALLOC macro parameter list. For the system to satisfy your DD name dynamic allocation request, the existing allocation must not be in use.

If the existing allocation with the specified DD name does not meet this requirement, or if the DD name is not associated with any of your program's existing allocations, the system fails the request and returns an error reason code in the SVC 99 parameter list.

In general terms, you can access files in C/C++ using two methods. One uses the traditional DD name format; the other allows the actual filename to be specified directly.

If the existing allocation meets the requirements, the system assigns it the in-use attribute, and the request has been satisfied. If the existing allocation is a member of a concatenated group, all members of the group are assigned the in-use attribute, so the entire group has been allocated.

Verb Code X'07' - Request for Information Retrieval: Dynamic information retrieval provides you with information about your current allocation environment. Verb code 07 allows you to request information such as the following:

- dataset name
- DD name
- pathname
- member name
- dataset organization
- status
- normal disposition
- abnormal disposition

Full details of the text units that need to be coded can be found in the IBM manuals referred to at the end of this article.

THE REAL WORLD

Now that I have covered all the basics of dynamic allocation, it's time to put this information to use in the real world! All of you high-level language programmers who've stayed with me this far can now see your reward!

In Figure 5, I present a fully working assembler subroutine that will dynamically allocate datasets for you and associate the DD name with a file description in your COBOL or PL/1 program. You can then OPEN, READ, CLOSE, and deallocate this file as your program dictates.

This process would be most useful when you have a list of files to read (or write) in sequence but the number of datasets to be handled is large. The overhead of coding such JCL would be excessive. This is an ideal situation for dynamic allocation. Remember there are no restrictions on the datasets you can allocate — they can equally be VSAM clusters, VSAM pathnames, as well as QSAM files or members of a PDS. I used this facility to build a VSAM key-scanning program to check for potential Year 2000 compliance issues with dates in keys.

A COBOL program could contain a "dummy" SELECT statement for a non-existent JCL DD name (e.g., FILEIN). Before OPENing FILEIN for input, the program invokes our subroutine (called DYNALLOC) with the sample code shown in Figure 6.

The code to check the error-code field is not shown, but basically, this shows the fundamental way this facility could be used. The actual DYNALLOC subroutine is more interesting.

Whereas all the subroutine does is allocate the specified dataset to the specified DD name with DISP=SHR (file must be catalogued) and deallocate it when requested, the error handling shows use of the RBX. In its simplest form, the RBX has been set up to send messages to the end user (in this case a batch job via WTO). IEFBD476 can also be used to extract messages from DYNALLOC, but this involves considerably more effort. More details can be found in the manuals referenced in the conclusion.

An alternate way to develop a DYNALLOC subroutine would be to build all the SVC 99 parameters in your COBOL or PL/1 program and pass the address of this block to a simple assembler routine that

Figure 5: Full Working Version of Subroutine 'DYNALLOC'

```
DYNALLOC    CSECT
            SAVE   (14,12)
            BALR   12,0
            USING  *,12
            ST     R13,SAVEAREA+4
            LA     13,SAVEAREA
            B      STARTUP
            SPACE
SAVEAREA    DC     18F'0'
            SPACE
*-------------------------------------------------------------------*
*         REGISTER EQUATEs                                          *
*-------------------------------------------------------------------*
            SPACE
R0          EQU    0
R1          EQU    1
R2          EQU    2
R3          EQU    3
R4          EQU    4
R5          EQU    5
R6          EQU    6
R7          EQU    7
R8          EQU    8
R9          EQU    9
R10         EQU    10
R11         EQU    11
R12         EQU    12
R13         EQU    13
R14         EQU    14
R15         EQU    15
            SPACE
*-------------------------------------------------------------------*
*         ADDRESS PARAMETER LIST AND BUILD SVC 99 PARAMETER BLOCK   *
*-------------------------------------------------------------------*
            SPACE
STARTUP     DS     0H
            LM     R2,R3,0(R1)             ADDRESS PARAMETERS PASSED IN
            USING  PARMS,R2
            CLI    REQUEST,C'1'            IS IT ALLOC
            BE     ALLOC                   YES, PROCESS
            CLI    REQUEST,C'2'            IS IT DEALLOC ?
            BE     DEALLOC                 YES, PROCESS
            LA     R15,2000                NO, ERROR - SET RETURN-CODE
            B      EXIT
*
ALLOC       DS     0H
            MVI    RBVERB,X'01'            SET REQUEST VERB
            MVC    TEXTU1+6(8),DDNAME      SETUP DDNAME
            MVC    TEXTU2+6(44),DSNAME     SETUP DSNAME
            CLC    DSMEMBER,=CL8' '        IS THERE A MEMBER NAME ?
            BNE    MEMBER                  YES, SETUP CORRECT SVC 99 PARMS
            MVC    RBTEXTA(4),=A(TEXTLST2) ADDRESS CORRECT TEXT UNIT LIST
            B      SVC99
*
MEMBER      DS     0H
            MVC    TEXTU3+6(8),DSMEMBER    SETUP MEMBER NAME
            MVC    RBTEXTA(4),=A(TEXTLST1) ADDRESS CORRECT TEXT UNIT LIST
            B      SVC99
*
DEALLOC     DS     0H
            MVI    RBVERB,X'02'            SET REQUEST VERB
            MVC    TEXTU1+6(8),DDNAME      SETUP DDNAME
            MVC    RBTEXTA(4),=A(TEXTLST3) ADDRESS CORRECT TEXT UNIT LIST
            B      SVC99
*
SVC99       DS     0H
            LA     R1,RBA
            DYNALLOC
            MVC    ERRCODE,RBERROR         RETURN ERROR CODE INFO
            LTR    R15,R15                 GOOD SVC 99 CALL ?
            BNZ    EXIT                    NO, TELL THE CALLER
            CLI    REQUEST,C'1'            WAS IT AN ALLOC REQUEST
            BNE    EXIT                    NO, BACK TO CALLER
            MVC    40(8,R3),DDNAME         SET DDNAME IN DCB
*
EXIT        DS     0H
            L      R13,4(R13)
            RETURN (14,12),RC=(15)
            EJECT
*
*-------------------------------------------------------------------*
*           WORKING STORAGE AREA                                    *
*-------------------------------------------------------------------*
```

(continued on next page)

has only a handful of instructions. Basically, the assembler code loads the parameter address into R1, invokes DYNALLOC and returns the return code in R15. I'll leave it up to your own ingenuity to build the SVC 99 request block and text units in a high-level language. Sample layouts of the PL/1 definitions can be found as comments at the end of the IEFZB4D0 copybook. Finally, it's also useful to note that, strictly speaking, deallocation is unnecessary as the DD name/dataset name combination will be automatically freed at job step termination. However, as one of the reasons why we would use dynamic allocation is to hold the resource for as short a time as possible, then good practice dictates we free our resources as soon as we have finished with them. Alternatively, we could code FREE=CLOSE when dynamically allocating our datasets in the first place, and then, this task will be performed automatically for us! There is more than one way to skin the proverbial cat!

C/C++ PROGRAMS

Probably, not many mainframe programmers have experienced the joys of writing code in C/C++ for the MVS system as yet! While everything I said before about calling an assembler routine to allocate/deallocate files holds true for C/C++ programs, these languages come with a user-friendly interface to dynamic allocation.

In general terms, you can access files in C/C++ using two methods. One uses the traditional DD name format; the other allows the actual filename to be specified directly. C/C++ internally calls dynamic allocation to "attach" the file (with a system-generated DD name) and open it in one function call (fopen). That's the easy way! C/C++ also comes with MVS extensions to the ANSI standard to these languages that provide functions called *dyninit()*, *dynalloc()* and *dynfree()*. These take parameter structures that define the SVC parameter list. So, with C/C++ you can use dynamic allocation directly in your program code, unlike COBOL or PL/1. To dynamically allocate a dataseton MVS in C/C++, you should follow these steps:

1. Invoke dyninit() with a variable of type __dyn_t.
2. Assign values to the appropriate fields in the variable that will satisfy the SVC 99 request.
3. Invoke dynalloc() with this variable.

(Figure 5 continued from page 19)

```
*
RBA         DC     0F'0',X'80',AL3(RB)
*
RB          DC     AL1(20)                 Length of Request Block
RBVERB      DC     X'01'                   Verb - 01 for Alloc, 02 for Free
RBFLAGS1    DC     AL2(0)                  Flags 1 - initially set to zero
S99NOMNT    EQU    X'2000'                 Do not mount volumes
S99NOMIG    EQU    X'0100'                 Do not recall migrated file
RBERROR     DC     AL2(0)                  Error Code
RBINFO      DC     AL2(0)                  Information Code
RBTEXTA     DC     A(TEXTLST1)             Address of text unit list
RBEXTNA     DC     A(RBX)                  ADDRESS OF RB EXTENSION ( ZERO )
RBFLAGS2    DC     A(0)                    Flags 2 - initially set to zero
*
RBX         DS     0F
RBXID       DC     CL6'S99RBX'             RBX IDENTIFIER
RBXVER      DC     XL1'01'                 RBX VERSION NUMBER
RBXOPTS     DC     XL1'84'                 ISSUE ERROR MSG VIA WTO    <====
RBXSUBP     DC     XL1'00'                 NO SUBPOOL SPECIFIED
RBXSKEY     DC     XL1'00'                 NO STORAGE KEY SPECIFIED
RBXMSGL     DC     XL1'08'                 MIMIMUM SEVERITY - SEVERE <====
RBXNMSG     DC     XL1'00'                 NUMBER MSG BLOCKS RETURNED - 0
RBXCPPL     DC     F'0'                    TSO COMMAND PROCESSOR - NOT USED
RBXRES1     DC     XL2'0000'               RESERVED
RBXERCO     DC     XL1'00'                 MSG PROCESSING REASON CODE
RBXERCF     DC     XL1'00'                 MSG BLOCK FREEING REASON CODE
RBXWRC      DC     F'0'                    WTO RETURN CODE
RBXMSGP     DC     F'0'                    ADDRESS OF MSG POINTERS
RBXERR      DC     XL2'0000'               INFORMATION RETRIEVAL ERR CODE
RBXINFO     DC     XL2'0000'               INFORMATION RETRIEVAL INFO CODE
RBXSMSRC    DC     F'0'                    SMS REASON CODE
*
TEXTLST1    DC     A(TEXTU1),A(TEXTU2),A(TEXTU3),X'80',AL3(TEXTU4)
TEXTLST2    DC     A(TEXTU1),A(TEXTU2),X'80',AL3(TEXTU4)
TEXTLST3    DC     A(TEXTU1),X'80',AL3(TEXTU7)
*
TEXTU1      DC     X'0001',X'0001',X'0008',CL8'SYSUT1'       DDNAME=
TEXTU2      DC     X'0002',X'0001',X'002C',CL44' '           DSN=
TEXTU3      DC     X'0003',X'0001',X'0008',CL8' '            MEMBER NAME
TEXTU4      DC     X'0004',X'0001',X'0001',X'08'             DISP=SHR
*
TEXTU7      DC     X'0007',X'0000'                           FREE
*
PARMS       DSECT
REQUEST     DS     C
DDNAME      DS     CL8
DSNAME      DS     CL44
DSMEMBER    DS     CL8
ERRCODE     DS     XL2
*
            END
```

Figure 6: Sample COBOL Code to Invoke 'DYNALLOC'

```
       WORKING-STORAGE SECTION.
...
       01    WS-DYNAPARM.
             03  WS-FUNCTION                    PIC 9.
                 88  ALLOCATE                   VALUE 1
                 88  DEALLOCATE                 VALUE 2.
             03  WS-DDNAME                      PIC X(8)  VALUE 'FILEIN'.
             03  WS-DSNAME                      PIC X(44).
             03  WS-MEMBER                      PIC X(8)  VALUE SPACES.
             03  WS-ERROR-CODE                  PIC 9(4)  COMP.
...
       PROCEDURE DIVISION.
...
             MOVE   1            TO WS-FUNCTION.
             MOVE   DDNAME       TO WS-DDNAME.
             MOVE   DSNAME       TO WS-DSNAME.
             CALL   'DYNALLOC'   USING WS-DYNAPARM INPUT-FILE.
             IF     RETURN-CODE NOT EQUAL ZERO
                    PERFORM ERROR-PROCESSING
                    GOTO EXIT-OUT.
...
             OPEN INPUT INPUT-FILE.
             PERFORM READ-INPUT-FILE UNTIL INPUT-FILE-EOF.
             CLOSE INPUT-FILE.
...
             MOVE   2            TO WS-FUNCTION.
             CALL   'DYNALLOC'   USING WS-DYNAPARM INPUT-FILE.
             IF     RETURN-CODE NOT EQUAL ZERO
                    PERFORM ERROR-PROCESSING
                    GOTO EXIT-OUT.
```

The C/C++ reference manuals provide all the gory details on how to use these facilities along with some good examples.

SUMMARY

I hope this two-part tutorial on dynamic allocation has opened up a new area of MVS that you never knew existed. It shows a flexible method of allocating resources to a program for use in specialized applications. While it is true that the issues involved are quite complex, the components (such as control blocks) presented here can be used as a basis for solutions in your environment. Enjoy!

REFERENCES

When working with dynamic allocation, the IBM-supplied texts are essential reference materials for building request blocks and text units, as well as providing full details of all possible return codes, error codes, and information codes. The manual I used as a reference for this tutorial is *MVS/ESA Programming: Authorized Assembler Services Guide*, MVS/ESA System Product: JES2 Version 5, JES3 Version 5, Document Number GC28-1467-02.

ISPF help facility, Appendix A gives all the return and error code meanings and is useful as a quick reference.

Also, see *IBM C/C++ for MVS/ESA Library Reference*, Version 3, Release 1, Document Number SC09-1995-00.

All of the source code snippets and figures accompanying this article are available in file FEB98001.EX1 of the TECHSUPT LIB of the NaSPA Internet server. That should save you some typing if you wish to get started with MVS dynamic allocation! ts

NaSPA member Michael H. Carroll has 19 years of experience in the data processing industry. He's worked in both manufacturing and financial businesses, performing systems and applications programming for mainframe and client/server environments. Michael is a consultant currently working on Y2K projects for a large financial institution in Ireland.