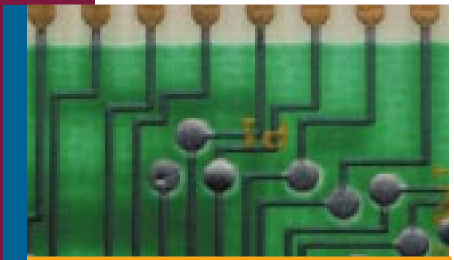

BY MICHAEL H. CARROLL

# MVS Dynamic Allocation: Part I

MVS dynamic allocation presents a way to allocate, or assign, I/O resources to an executing program and allows you to tailor your device allocations based on input to your program.

AS programmers writing programs in the MVS mainframe environment with languages such as COBOL, PL/I or Assembler, we are familiar with what is required to execute the code we have produced. We typically code some JCL to invoke the program, supplying all the needed resources to the program by coding DD statements to allocate files for input and output. This has been the age-old way of doing things and indeed, the easiest and most flexible method in most cases. There has also been another way to assign resources to an executing program that has been part of MVS since the early years. Some systems programmers may be aware of this facility, but not many programming personnel in general know how to use this alternative method, MVS dynamic allocation.

## GENERAL CONCEPTS

Allocation is the process by which the system assigns, or allocates, I/O resources to your job. An I/O resource is a DD name-dataset combination with any associated volume and device information. Deallocation is the process by which the system releases or deallocates I/O resources that were allocated to your job.

There are two basic types of allocation: job step allocation and dynamic allocation. The two types allocate resources at different points in program processing. Job step allocation assigns resources to your program before your program runs, and dynamic allocation assigns resources to your program while it is running. The needs of your program determine which type of allocation you should use.

When using job step allocation, you request I/O resources through JCL. The system allocates those I/O resources before your program runs as part of initiating the job step and deallocates resources after your program runs as part of job step termination. This type of allocation ensures that the resources you request are available before your program runs and throughout program execution. The only circumstance in which you can influence how long a job step keeps a resource allocated is when using FREE=CLOSE to release the resource before job step completion.

When using dynamic allocation, you request I/O resources by coding the DYNALLOC macro (which invokes SVC 99) and filling in the fields of the SVC 99 parameter list. The system allocates and deallocates those I/O resources while your program is running. Dynamic allocation also allows you to request information about your allocation environment and to deallocate or modify characteristics of your allocation environment that were acquired either dynamically or through JCL.

Dynamic allocation allows you to tailor your device allocations based on input to your program. You can design your program to dynamically allocate only those devices that are necessary in a particular programming path, rather than allocate all possible device requirements before your program runs.

Dynamic allocation also allows you to use common resources more efficiently. When there is high contention for a resource, dynamic allocation allows you to acquire an I/O resource just before you need it and to release it as soon as you no longer need it, so that your program holds the resource for a shorter period of time.

Figure 1: When To Use Job Step and Dynamic Allocation

| WHEN... | CHOOSE: |
|---|---|
| You need data to be available for the duration of your program. | JCL |
| You need to know that the program has access to all data before execution begins. | JCL |
| Your data requirements are constant for all program conditions. | JCL |
| You need data only in certain paths of program processing. | Dynamic Allocation |
| Your program can wait or fail if the data is not available when you issue the DYNALLOC macro. | Dynamic Allocation |
| Your program is a long-running job or server that could cause contention for system resources. | Dynamic Allocation |

Figure 2: Dynamic Allocation Primary Functions

| FUNCTION | PURPOSE |
|---|---|
| Dsname allocation | Dynamically assigns a dataset to a job by its dataset name. |
| Deallocation | Dynamically releases resources assigned to a job through JCL or dynamic allocation. |
| DDname allocation | Dynamically reuses a not-in-use dataset. |
| Concatenation | Logically associates allocated datasets. |
| Deconcatenation | Logically disassociates concatenated datasets. |
| Information Retrieval | Retrieves information about the allocation environment. |

## WHICH METHOD IS BEST?

When should we think of using dynamic allocation? Why should we even consider this option when the job step allocation method has always worked for us? Figure 1 outlines the IBM recommended allocation method to use based on the needs of your program. This is not the end of the story, of course. The DYNALLOC macro mentioned previously is an assembler macro, which implies that dynamic allocation is only available to assembler programmers. Before all high-level language programmers' eyes glaze over and they skip to the next article, let me say that while it's true that DYNALLOC can only be invoked by assembly language, it's possible to write assembler routines that can be called from languages such as COBOL and PL/1. These subroutines can "tap into" the power of dynamic allocation. I'll present one such routine in Part II of this series.

There are specific instances where dynamic allocation should be avoided. These concern system-related activities, which should only be of direct interest to systems programmers:

- programs running in cross-memory mode
- programs running under an Interruption Request Block (IRB)
- installation exits that get control duringthe start of the Job Entry Subsystem (JES)
- multitasking programs in which one task issues the DYNALLOC macro and other tasks issue the DYNALLOC or OPEN macros
- MVS command installation exits

Most dynamic allocation functions are useful in either a batch environment or an interactive environment such as TSO/E. Where there are specific rules governing the batch or interactive environment, I'll outline them as we go along.

There are six primary functions provided by MVS dynamic allocation. These are outlined in Figure 2. This article will concentrate on dsname allocation and deallocation.

## DSNAME ALLOCATION

The major function performed by dynamic allocation, and the function most often requested, is that of dynamically allocating a dataset according to its name (dsname or pathname). Dynamic allocation by dsname or pathname is equivalent to dataset or file allocation during job step initiation except that the resource is allocated as your program runs.

Before using dsname or pathname allocation, you should ensure that the service you need is available through dynamic allocation. You can request most of the JCL facilities that you can code in a DD statement, such as data set disposition, volume label information, expiration date, and SYSOUT destination, by specifying the appropriate text units in the parameter list. This will be explained later. However, some JCL facilities do not have dynamic allocation equivalents. These are summarized in Figure 3.

When you invoke the DYNALLOC macro to perform dsname dynamic allocation, an "allocation environment" already exists for your request. It consists of the allocation requests made through your JCL or earlier dynamic allocations that have not yet been deallocated. The system considers these resources to be existing allocations and goes to them first to fill your dynamic allocation requests.

Dynamic allocation cannot satisfy a dsname allocation request that is in conflict with your existing allocation environment. Environmental conflicts can cause your request to fail when your dsname allocation request specifies the following:

- a DD name that is associated with an existing allocation that is in use
- a DD name that is associated with a group of concatenated data sets defined as permanently concatenated (explained in Part II)
- a new non-temporary data set with the same dsname as an existing allocation
- an existing data set (by specifying a disposition of OLD or SHR) that is not permanently allocated, not in-use and has a disposition of DELETE. A data set with these characteristics might be deleted before your program requests it.

Figure 3: JCL DD Statement Facilities Not Supported by Dynamic Allocation

| | | |
|---|---|---|
| Restricted DDnames | JOBCAT, STEPCAT, JOBLIB and STEPLIB | |
| Keyword Parameters | CHKPT, DDNAME, DLM and DSID | |
| Positional Parameters | *, DATA and DYNAM | |
| Selected Sub-parameters of Keywords | **Keyword** | **Sub-parameter Not Supported** |
| | DCB | reference to DD name of a previous step |
| | | CYLOFL |
| | | NTM |
| | | RKP |
| | DISP | PASS specification |
| | DSN | reference to DD name (as in *.ddname) |
| | | ISAM area name |
| | SPACE | ABSTR specification |
| | UNIT | AFF |
| | VOLUME | RETAIN specification |
| | | REF=DD name |

You can avoid allocation of the incorrect data set in the following ways:

- Specify the volume and unit where the data set is to be allocated unless it is catalogued.
- Free existing allocations for the same data set name using dynamic deallocation.
- Use a unique data set name.

All this may seem confusing, but the simple rule to remember is that whatever you would do in JCL, you can, with few exceptions, do with dynamic allocation.

## DEALLOCATION

Deallocation releases resources allocated to your program. You can dynamically deallocate resources that were allocated either dynamically or through JCL, unless the data set is open or a member of an open concatenated group.

In either case, the data set is not deallocated. Releasing a data set results in the following:

- You can use the DD name in subsequent dynamic allocation requests.
- The system processes the dataset disposition.
- Other jobs can use the dataset except when the system has deleted it as part of disposition processing — e.g., DISP=(OLD,DELETE).
- The system frees the unit to which the dataset was allocated if the unit is not needed for subsequent job steps.
- The system releases the volumes on which the dataset was allocated if the volume is not needed for subsequent job steps.

Deallocation can be performed by dsname or DD name. If only dsname is used, then all DD names with that dsname are deallocated. DD name-only deallocations free the associated dsname. Other DD names associated with that dsname are not deallocated.

## THE NITTY-GRITTY

Now that we have outlined the theory, let's put some flesh and bones on this in a practical way. The DYNALLOC macro usually resides on SYS1.MACLIB (MVS-supplied macro library). If you look at this macro on your system, you will see that apart from some comments, it contains one line that generates a line of assembly code, namely an SVC 99 call. SVCs are system-supplied supervisor calls that provide access to MVS services. SVC 99 is the MVS dynamic allocation supervisor call. The parameters that control DYNALLOC must be supplied via Register 1. Figure 4 shows the layout of the SVC 99 parameter list. The major elements that we wish to use at this point are presented as follows.

## THE REQUEST BLOCK

Register 1 must point to a full-word that contains the address of the DYNALLOC Request Block (RB). The high-order bit of the RB must be set to 1.

The RB specifies the dynamic allocation function being requested. The major fields of immediate interest and their functions are as follows:

1. The length field always specifies a value of 20.

2. The verb code identifies the dynamic function being requested. The valid values are:

   X'01' - Request for dynamic allocation.
   X'02' - Request for deallocation.
   X'03' - Request for concatenation.
   X'04' - Request for deconcatenation.
   X'05' - Request for removing in-use attribute.
   X'06' - Request for DD name allocation.
   X'07' - Request for information retrieval.

3. Flags 1 field — for now we set this to binary zeroes.

4. Error/Info code bytes (full-word) where DYNALLOC returns error information.

5. Address of text units that outline the specific nature of our request.

6. All other fields initially contain binary zeroes.

To assist programmers, IBM has supplied a mapping DSECT copybook (IEFZB4D0) to give symbolic names to the fields in the RB and other structures. The easiest way to use this facility to allocate a block of storage in your assembler module is to load its address into a register and map the area with the USING statement. See Figure 5. I

Figure 4: DYNALLOC Parameter Blocks

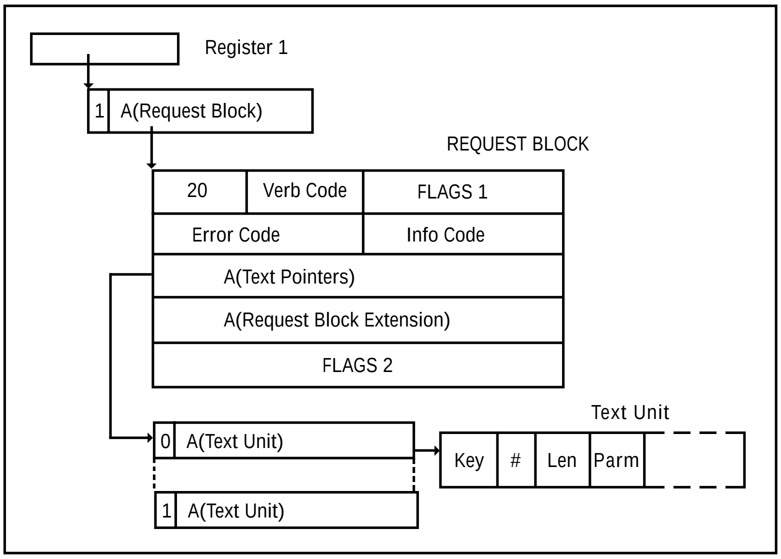

Figure 5: Sample Use of Copybook IEFZB4D0 and Request Block Definition

```
LA        R6,RB
          USING  S99RB,R6
          ...
          LA     R1,RBA
          DYNALLOC

          ...


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
RBERROR   DC      AL2(0),AL2(0)               Error code, 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
```

personally find it easier to assign a name to each field in the RB and initialize them to the values I wish to use. The particular method you use is a matter of personal choice.

The basic process for invoking DYNALLOC is setting the function verb in the RB to X'01' for allocate or X'02' for deallocate (RBVERB in Figure 5 or field S99VERB if using the IEFZB4D0 copybook), loading the RB pointer into R1 and invoking DYNALLOC.

## THE TEXT UNIT

In the RB, there is an address of a list of text units (RBTEXTA in Figure 5) which points to a list of addresses that comprise the parameters that allocate or deallocate use. These text unit(s) provide information such as dataset name, disposition, DD name, etc. The list of text unit addresses has the form specified in Figure 6. Each address in the list points at a text unit parameter. The last address in the list must have the high-order bit set to one (signifies end-of-list). There are a considerable number of text units, basically one for almost every type of parameter you can use in a JCL statement. The most common ones are listed in Figure 7 and the deallocation units are listed in Figure 8. The IBM manuals outline the complete list (see references at the end of this article). In Figure 6 you will notice that I've logically grouped a number of text units together. The address of TEXTLST1 is specified in Figure 5 (the RB). The list in Figure 6 points to three text units, a DD name declaration, a data set name definition (XYZ.DSN) and a disposition of SHR. This equates to the following identical JCL statement:

```
//SYSUT1    DD   DSN=XYZ.DSN,DISP=SHR
```

Simple, isn't it? To deallocate this file, all we need to do it move the address of TEXTLST2 into the RBTEXTA field, set the RBVERB field to X'02', load address of RB into R1 and invoke DYNALLOC.

The form of deallocate specified here is the DD name method and is probably the most common way to deallocate data sets using the dynamic facility. The copybook IEFZB4D0 contains a DSECT (S99TUNIT) for text unit layouts, also. Again, I find direct storage definitions (outlined in Figure 6) easier to use. Each text unit is a variable length block that consists of the following fields:

**Bytes 1 to 2** - hexadecimal number representing the text unit key (values specified in Figure 7).

**Bytes 3 to 4** - hexadecimal number specifying the number of length and parameter combinations in the text unit. For example, in Figure 6 TEXTU1 has one pair of length/ parameter values after this field.

**Byte 5 onwards** - two-byte hexadecimal value representing length of following parameter e.g., 8, 44, etc.; variable field representing actual parameter value e.g., DD name, dsname, etc.

Note that if there are no additional parameters, bytes 3 and 4 will be zero (e.g., TEXTU7 in Figure 6).

All the text unit layouts are documented in the relevant IBM manuals. To allocate and deallocate a dataset, then, the text units and RB described here are all that are required. Text units do not have to occupy storage adjacent to each other. The text units are accessed via the address of the list supplied in the RB (field RBTEXTA). Additionally, dataset names supplied in TEXTU2 (text unit field = X'0002') do not have to define the full 44 bytes for the field. I tend to use the full definition, so I can reuse the text unit for multiple dataset allocations. As long as the dsname moved into the text unit is space padded, then SVC 99 will act on the request. Low values or binary zeroes should not be used for padding this field.

## NEXT TIME

Part II will examine the other dynamic allocation functions, explain how we handle errors, present some real-world

Figure 6: Example of Text Unit Definitions

```
TEXTLST1  DC     A(TEXTU1),A(TEXTU2),X'80',AL3(TEXTU4)
TEXTLST2  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'XYZ.DSN'   DSN=
TEXTU4    DC     X'0004',X'0001',X'0001',X'08'           DISP=SHR
*
TEXTU7    DC     X'0007',X'0000'                         FREE
*
```

Figure 7: Sample Text Unit Keys For Dsname Allocation

| Hex Text Unit Key | Dsname Allocation Function |
|---|---|
| 0001 | Associates a DD name with an allocation request. |
| 0002 | Names the data set to be allocated. |
| 0003 | Specifies data set number or relative generation number. |
| 0004 | Specifies the data set status. |
| 0005 | Specifies the data set's normal disposition. |
| 0006 | Specifies the data set's conditional disposition. |
| 0007 | Specifies the space allocation in tracks. |
| 0008 | Specifies the space allocation in cylinders. |
| 0009 | Specifies the average data block length. |
| 000A | Specifies a primary space quantity. |
| 000B | Specifies a secondary space quantity. |
| 000C | Specifies the number of PDS directory blocks. |
| 000D | Deletes unused space at data set closure. |
| 000E | Ensures a specific allocated space format. |
| 000F | Specifies space allocation in whole cylinders. |
| 0010 | Specifies volume serial numbers. |
| 0011 | Specifies the private volume use attribute. |
| 0012 | Specifies the volume sequence number processing. |
| 0013 | Specifies the data set's volume count. |
| 0014 | Specifies volume reference to a catalogued data set. |
| 0015 | Describes the unit specification. |

Figure 8: Sample Text Unit Keys For Deallocation

| Hex Text Unit Key | Dsname Allocation Function |
|---|---|
| 0001 | Specifies the DD name to be deallocated. |
| 0002 | Names the data set to be deallocated. |
| 0003 | Specifies the PDS member to be deallocated. |
| 0005 | Specifies an overriding disposition for the dataset to be deallocated. |
| 0007 | Specifies deallocation even if the resource has the permanently allocated attribute. |

examples of how to use these facilities, and more importantly, how to exploit them. Join me next time!

## REFERENCES

*MVS/ESA Programming: Authorized Assembler Services Guide*, MVS/ESA System Product: *JES2 Version 5*, *JES3 Version 5*. Document Number GC28-1467-02

*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.*