VSE Data Spaces: Part I

BY LEO J. LANGEVIN

For some time now, people have been asking me about data spaces, virtual disks, access registers, and all of the other really great things that VSE/ESA offers to sites that are running in ESA mode. Although you can run 31-bit programs in VSE/ESA version 1 in XA-mode, you need access registers in order to use VSE/ESA version 2, and thus use data spaces.

WHAT IS A DATA SPACE?

A data space is an area of storage that is an extension of the operating virtual size where data can be added, modified, deleted, and retrieved. However, a data space is not the same as GETVIS, where an address is retrieved and can even be branched to for execution. A data space can only contain data. If you store a program in a data space, it will be treated as data and cannot be executed unless it is copied into an area where execution is allowed.

Each data space defined is unique and is not part of another data space. You cannot write past the area defined to that data space because any address beyond the limit of a defined data space doesn't exist and you will therefore encounter an addressing exception. Like GETVIS, you can request that the data space be extended or that a block of data space storage be deleted, causing it to become non-addressable/usable.

Data spaces start in the 24-bit area and can extend through the 31-bit area. This is important to note so that you use 31-bit programming in order to correctly access the desired area of storage. If you don't, then you may be pointing to the wrong area!

THE SYSDEF JCL STATEMENT

First, you need to tell VSE that you will be using data spaces. Then you will need to define a general pool that the system can use. The reason for this is that when you define a data space, the system will see if there is sufficient space left in that pool to support that request. You define the *total* storage that can be used. You define the pool of storage by using the SYSDEF JCL parameter, as shown below:

```
SYSDEF DSPACE
         [,DSIZE=(nKIMM)i
         [,MAX=nl[,PARTMAX=ml
         [,COMMAX=kl
         [,DFSIZE=fnKlffhm)l
```

Following are the options for the SYSDEF JCL command:

- **DSIZE:** the total size of the data space pool. Once allocated, it will not be available for dynamic partitions, so be careful.
- **MAX:** the maximum number of data spaces. The default value is 256.
- **PARTMAX:** limits the number of data spaces that any single partition can create. The default value is 16.
- **COMMAX:** defines the maximum number of SCOPE=COMMON data spaces that may exist at any one time. Virtual disks are common and need to be included in this total. The default is 5, and the maximum amount is 253.
- **DFSIZE:** the default size for the definition of data spaces when no size is provided within the application program. The size must be in multiples of 32KB. If no value is provided, then the system will default to 960KB.

A data space is an area of storage that is an extension of the operating virtual size where data can be added, modified, deleted, and retrieved. However, a data space is not the same as GETVIS, where an address is retrieved and can even be branched to for execution.

LAYOUT OF VSE WHEN USING DATA SPACES

Figure 1 shows a generic layout of the VSE/ESA operating system with three data spaces defined. The 24-bit shared area includes the SVA, supervisor, etc. Notice that the data spaces start at the beginning of storage (address = X'00000000') and can be expanded up to the maximum of 2GB of virtual storage each.

VIRTUAL STORAGE CONSIDERATIONS

Data spaces will provide your operating system horizontal growth similar to dynamic address spaces. Both are similar in that they do not exist until they are dynamically defined, and once defined they allocate necessary resources that cannot be used by another requester until those resources have been released.

For example, let's say that you have defined a system that has 80MB of virtual storage. After all of the address spaces have been defined, whatever free space is left can be used for the allocation of data spaces and dynamic partition spaces. If you end up with 40MB of free virtual storage and then define a data space pool of 30MB by using the SYSDEF command there will only be 10MB of virtual storage left for the definition of a dynamic partition. If there is insufficient space available, then the allocation will fail. Therefore, carefully define the amount of VSIZE that your operating system will require.

Once the required amount of free storage is available from the VPOOL definition, and a pool of data space is defined by using the SYSDEF request, you can have your application program allocate an area for defining a data space.

As shown in Figure 1, each data space lies in its own unique area. Each data space has its own access register token to allow access from any partition. No two data spaces share the same area. In contrast, you can define more than one static partition to exist in a single address space. Data spaces do not share this design.

DEFINING VIRTUAL DISKS: EXAMPLES OF DATA SPACES

Virtual disks are data spaces the operating system uses to simulate real disk drives without the I/O wait. This means I/O at the speed of memory! Of course, there may still be some real I/O resulting from using a virtual disk, especially if your system starts paging due to the lack of sufficient real memory. However, paging I/O is approximately 800 percent more efficient with VSE/ESA because of the way that paging requests are now being chained.

To define a virtual disk, you must first have one or more PUB definitions. This is part of your standard IPL procedure. For example, if you wanted to define a virtual disk address of X'600', you would need to have the following statement included in your IPL:

```
ADD 600,FBAV
```

FBAV is the device type for a virtual disk that will be emulating an FBA device.

The next step is to define a pool of storage by using the SYSDEF command. To reserve 6MB for this purpose you might use

```
SYSDEF DSPACE,DSIZE=6M
```

Finally, you would use the VDISK command to define a virtual disk. Let's say that you want to define a virtual disk to be used for storing the label area. In that case you might use

```
VDISK UNIT=600,BLKS=500,USAGE=DLA
```

Figure 1: Generic Layout of the VSE OS With Three Data Spaces Defined

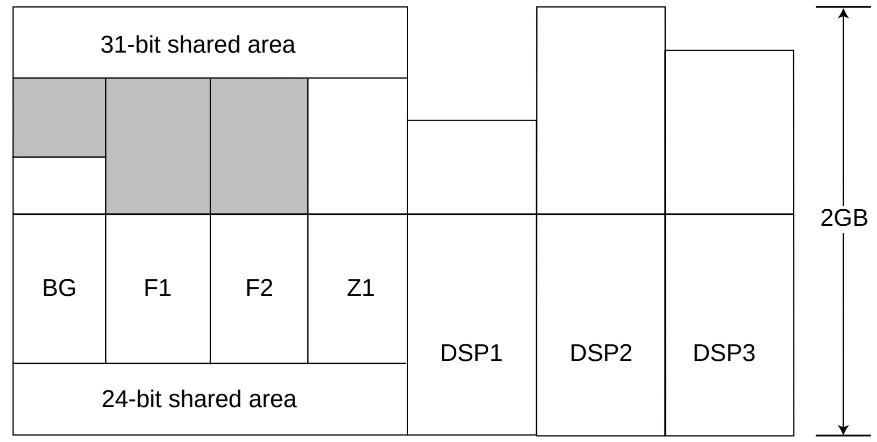
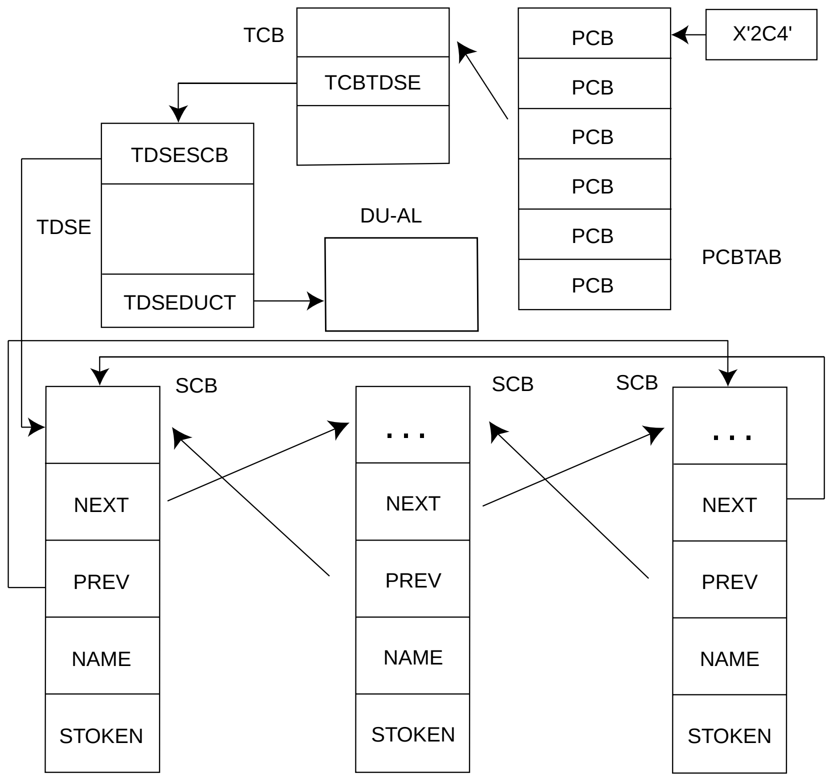


Figure 2: Control Block ationships for Data Spaces



After you have defined your virtual disk, it is now ready for use. To load the label area, you may want the next statement to be

```
OPTION STDLABEL
// DLBL ...
```

You can expand and contract the data space pool of virtual storage by using the SYSDEF command at any time. If a data space is in use, then you will not be able to shrink the data space pool smaller than the size required to support the active data space.

You can release the space that a virtual disk has allocated by resetting the size to zero and issuing a DVCDN request to that address. However, in the label area example this is not a valid request, since eliminating the disk where a system file is located would cause the operating system to crash.

Let's say we had a second virtual disk (60 1) that we were no longer using and we wanted to release the storage. We could then issue the following:

```
DVCDN 600
VDISK UNIT=601,BLKS=0
```

A DATA SPACE INTERNALS PRIMER

VSE has implemented a version of data space access that is easy to use. As with any operating system structure there are control blocks and system pointers that are updated whenever a data space is defined. However, it is not necessary to use these pointers to use a data space. I am presenting these control blocks so that you can better understand how to perform functions that exceed the normal interface. I'll describe the process in reverse so that you will have a better understanding on how this all fits together. Afterward, I will provide a diagram that will make it even clearer.

As mentioned, every data space has a unique data space token assigned to it. This 8-byte token or field is the identifier for that specific data space. Once you acquire this token, you can use the ALESERV macro to get the ALET that is necessary to access the area within that data space.

The space token, or STOKEN, is maintained in the Space Control Block (SCB). By using the MAPDSCB macro, you can generate a layout of the SCB. Note that multiple SCBs are chained together. In one SCB is the pointer to the previous and the next SCB address.

The initial SCB is pointed to by the Task Data Space Element (TDSE). A layout of this field is generated by using the MAPTDSE macro. By using field TDSESCB you will be pointing to the initial SCB in the chain.

Every executing program has one or more tasks associated with it. Each task in the system is provided with a unique identifier which allows the system to associate that task with a control block which defines the important features and attributes of that task such as save areas, GETVIS control information, and other data. This control block is called the Task Control Block (TCB). The DSECT for the TCB can be created by using the MAPTCB macro. Use field TCBTDSE to point to the TDSE control block.

The TCB can be pointed to by accessing the Partition Control Block (PCB). The layout of the PCB is generated by using the MAPPCB macro. The specific PCB is part of a PCB table. The PCB table address is located in low core at X'24C'. The layout of low core can be generated by using the SGLOWC macro. The PCB is pointed to by accessing the partition's PIK and using this as a KEY to point to the correct PCB within the PCB table.

As you can see from Figure 2, it's fairly easy to get the names of the datasets that are defined to the system. In addition to obtaining these names, you can also get the associated STOKEN. The source code for a subroutine called DSPOPEN is available for download from the TECH SUPT library of the NaSCOM Internet server as filename MAR98001.ZIP. DSPOPEN will accept the name of any data space and return the STOKEN value. It performs the exact type of chaining that you see in Figure 2.

When you define a new data space, an SCB is inserted into the chain. After defining the space, you will need to add an ALET so that you can use the space. This means that the order of execution for defining a data space is DSPSERV followed by ALESERV. When you are done, use ALESERV followed by DSPSERV.

That's it for now. Next month, I will examine how to write you own application program that will define, extend, and delete data spaces. ts

NaSPA member Leo Langevin is the lead developer for NFS for VSE from Connectivity Systems. He has been involved with VSE Internals since its inception. He can be reached at leo@tcpip4vse.com.