An Inside Look at DCL

Guy Peleg
OpenVMS Ambassador

Part I – An Inside Look at DCL

This section describes various DCL process data structures.

The DCL process data structures are defined in [DCL]DCLDEF.SDL, and are allocated from P1 space in [SYS]SHELL.MAR.

The PRC Data Structure

The PRC data structure contains the process CLI work information. The R11 register always points to this data structure, and it is initialized in [DCL]INITIAL.MAR. The PRC data structure is located directly after the PRD data structure. The address of the beginning of this structure can be found by looking at CTL$AG_CLIDATA+8. The PRC data structure holds information that involves the process itself, and this data is maintained from image rundown to image rundown.

To access the PRC data structure, use the following commands:

```
SDA> READ SYS$LOADABLE_IMAGES:DCLDEF
SDA> EXAMINE CTL$AG_CLIDATA + 8
SDA> DEFINE  PRC=@.
SDA> FORMAT PRC/TYPE=PRC
```

Most of the information about DCL's internal structures is accessible through the use of various P1 space data cells.

DCL's data structures are defined by [DCL]DCLDEF.SDL and are described for SDA's use in SYS$LOADABLE_IMAGES:DCLDEF.STB

The PRD Data Structure – RMS Information

The PRD data structure contains RMS information for the process. This is the first data structure located at CTL$AG_CLIDATA in P1 space, and it contains the process FAB and NAM, as well as the RABs for input and output with the associated XABs. It is also contains file information for queueing the job log file to the job controller.

The WRK Data Structure

The WRK data structure contains the information about parsed commands. During the execution of DCL, the R10 register holds the address of this data structure. There is one copy for interactive level (with supervisor mode access) and possibly one other copy for the currently running image (with user mode access).

There is a possibility of having two WRK data structures at any given time. One structure always exists, and is allocated on the supervisor stack just before DCL reads each new

command in [DCL]COMMAND.MAR. The other structure is allocated from the user's virtual memory.

The supervisor stack copy holds the parsing information for commands parsed in user mode. PRC_L_SAVFP holds a pointer to this structure at all times.

The user mode copy holds the parsing information for commands that DCL is parsing for an image. This copy is allocated when CLI$DCL_PARSE is called without a command and prompting routines. If the routine is called without a command, it is assumed that the image is inquiring about the command that invoked it, and the parsing routines will use the supervisor mode copy.

The global cell CTL$AG_DCLPRSOWN points to the copy of the WRK data structure that the image is currently referring to. If no calls to CLI$ routines have been made, this cell is zero. This is also the case when there is no image currently running.

To access the image's WRK data structure, perform the following commands:

SDA> READ SYS$LOADABLE_IMAGES:DCLDEF
SDA> EXAMINE CTL$GL_DCLPRSOWN
SDA> DEFINE WRK= @.
SDA> FORMAT WRK/TYPE=WRK
To access DCL's WRK data structure, perform the following commands:

SDA> READ SYS$LOADABLE_IMAGES:DCLDEF
SDA> EXAMINE CTL$AG_CLIDATA+8
SDA> DEFINE PRC= @.
SDA> DEFINE WRK=PRC+PRC_L_SAVFP
SDA> FORMAT WRK/TYPE=WRK
The input and expansion buffer are located at WRK_G_INPBUF and WRK_G_BUFFER respectively (both are zero terminated string):

SDA> EXAMINE WRK+WRK_G_INPBUF;XX
SDA> EXAMINE WRK_WRK_G_BUFFER;XX
For most purposes, the expansion buffer is the one you want to look at, because if there is a continuation in the command, the input buffer is reused to obtain the continuation.

The Recall Buffer Data Structure

The following figure displays the contents of a recall buffer entry:

String n length	0	…
…String n …		
…		String n length

The items in the recall buffer entry appear as follows:
1. Byte containing 0
2. Byte containing the length
3. Variable length string (provided it is less than 255 bytes)
4. Length of the string

The buffer wraps as necessary.

The start address of the command recall buffer is denoted by PRC_G_COMMANDS and the length is denoted by PRC_C_CMDBUFSIZ (in bytes). The pointer to the address of the current command is PRC_L_RECALLPTR.

When using continuation lines, the first segment is placed in the buffer as though it were the entire command. DCL then obtains the next segment and concatenates it with the previous segment. During concatenation, the "-" character is removed from the previous string, and the two length bytes are updated.

For the next OpenVMS version, the recall buffer will be updated to support commands larger than 255 characters.

You can access the recall buffer using the following commands:

```
SDA> READ SYS$LOADABLE_IMAGES:DCLDEF       !GET DCL SYMBOLS
SDA> EXAMINE  CTL$AG_CLIDATA+8 !GET TO THE PRC
SDA> DEFINE PRC=@.
SDA> EXAMINE PRC+PRC_G_COMMANDS;XX
```

The pointer to where the next command (for example, the end of the last interactive command) will be is PRC_L_RECALLPTR. Note that the data continually circles the buffer.

A more automated way to look at the recall buffer is to use the CLUE command:

```
SDA> CLUE PROCESS/RECALL
```

File Handling Information

When a command is being read from the terminal or from a command procedure, it is being stored in the input buffer in the WRK structure. If the command exceeds the 255 characters limit, it is stored in the expansion buffer (also within the WRK structure).

There are two instances of inputting records. One instance is the READ command. Here, the buffer is allocated on stack at the time RMS is called. The other instance is reading input records from the terminal or from command procedures. In this case, the buffer has been allocated as part of the WRK data structure; however, it is still on the stack.

When a command procedure is called from another command procedure, DCL stacks the information on the current file and closes it. When the called file finishes executing, the file is closed, and then DCL unstacks the saved information and re-opens the calling file.
DCL has files that survive image rundown. These process-permanent files include SYS$INPUT, SYS$OUTPUT, SYS$COMMAND, and SYS$ERROR. Process-permanent files are also those files that are opened with the DCL OPEN command. For those files opened with the OPEN command, DCL creates a logical name of the user's specification containing the ESCAPE character (18 HEX) as the first word of the equivalence string (for example, byte #0 has 0, and byte #1 has 18).

When a file operation is done, (for example, a CLOSE, READ, or WRITE command), the following actions are performed:

- The logical name is translated
- The first word of the equivalence string is checked that it is 18 HEX
- The Internal File Identifier (IFI) is extracted
- The "alleged" IFI is checked with the following until a valid RAB is found:

    - The IFI is checked with the current SYS$INPUT and SYS$OUTPUT (IDF_W_INPIFI and IDF_W_OUTIFI of the current procedure level).
    - The IFI is checked with the SYS$INPUT and SYS$OUTPUT of procedure level 0
    - The IFI is checked with the process permanent file list formed by the OPEN command (PRC_L_PPFLIST is the list header

File commands are processed in [DCL]FILECMDS.MAR.

The Lexical Functions

This section describes the way lexical functions are handled within DCL.
Lexical function processing starts with a general code to parse and check the data type of the specified parameters, and then is dispatching to the specific code for the individual lexical function.

A scratch buffer is provided for the dispatching code. This buffer is actually the unused portion of the expansion buffer (WRK_L_EXPANDPTR – WRK_L_CHARPTR). Input for each lexical function is as follows:

R2/R3 - Descriptor of scratch area
P1(AP) - Descriptor of first parameter
P2(AP) - Descriptor of second parameter
Pn(AP) - Descriptor of nth parameter, up to MAX_ARGS

R1/R2 - Descriptor describing result (If R2 is zero, then R1 describes an integer. If R2 is nonzero, then R1/R2 describes a string).

Lexical function keywords:

Each function keeps its own table of legal keywords. Each lexical function is of a different structure and is used to check the user-entered parameters when a keyword is expected.

The following table lists files containing lexical function keyword lists. These files are referenced by DCL; however, they are not part of the DCL facility. Note that any keyword added to these tables is automatically supported by DCL.

Lexical function	File containing definition
F$GETDVI	[VMSLIB]DVITABLE.MAR
F$GETJPI	[VMSLIB]JPITABLE.MAR
F$GETQUI	[VMSLIB]QUITABLE.MAR
F$GETSYI	[VMSLIB]SYITABLE.MAR

F$GETSYI [SYS]SYSPARAM.MAR

The following table lists system services used by lexical functions:

Service	Lexical Function
DCL internal	F$CVSINTEGER
	F$CVUINTEGER
	F$DIRECTORY
	F$EDIT
	F$ELEMENT
	F$ENVIRONMENT
	F$EXTRACT
	F$INTEGER
	F$LENGTH
	F$LOCATE
	F$STRING
	F$TYPE
	F$VERIFY
LIB$CVT_(D)TIME	F$CVTIME
SYS$ASCTIM	F$TIME
SYS$FAO	F$FAO
SYS$GETDVI	F$GETDVI
SYS$GETJPI	F$GETJPI
	F$MODE
	F$PID
	F$PROCESS
	F$USER
SYS$GETMSG	F$MESSAGE
SYS$GETQUI	F$GETQUI
SYS$GETSYI	F$GETSYI
SYS$IDTOASC	F$IDENTIFIER
SYS$SETDFPROT	F$ENVIORNMENT
SYS$SETPRV	F$PRIVILEGE
	F$SETPRV
SYS$TRNLNM	F$LOGICAL
	F$TRNLNM
RMS	F$PARSE
	F$SEARCH
	F$FILE_ATTRIBUTES

Part II –New DCL Features
This section provides a brief overview of latest DCL enhancements.

OpenVMS V7.3-1, together with VMS731_DCL-V0100 TIMA kit, added some long-requested new DCL features:

The RECALL command:

The following features have been added to the recall buffer:

The /ALL qualifier returns all the commands starting with a specific string
For example, to return all commands starting with “SHO”, type the following command:

$ RECALL/ALL SHO

The /SEARCH qualifier searches the entire recall buffer for a specific string. For example:

$ RECALL/SEARCH dev
  1 show device dk
  2 dir sys$sysdevice:[000000]

New keywords in F$GETSYI:

The following keywords have been added to F$GETSYI:

TOTAL_PAGES, USED_PAGES, FREE_PAGES, and MODIFIED_PAGES.

These new keywords return information about memory usage; therefore, procedures can now make decisions based on the amount of available memory.

For example, you can specify to start an application only if at least 80MB of physical memory is available

New keywords in F$CVTIME:

The following keywords have been added to F$CVTIME to ease calculation of time relative to year. The new keywords are:

DAYOFYEAR, WEEKOFYEAR, HOUROFYEAR, and SECONDOFYEAR.