Introduction

This module presents the student with the use and mechanics of OpenVMS libraries. Students will be able to link against and produce object libraries, examine text libraries and produce help libraries. Students will be exposed to the procedural support available from OpenVMS, through Run-Time Libraries and System Services. Students will be able to respond to errors in procedure calls by processing Condition Values. Students will learn to construct simple DCL commands.

Objectives

Students should be able to:

- Identify the use and types of libraries available under OpenVMS.
- Create libraries and link programs using them.
- Differentiate the purpose of Utility, Run-Time Library, and System Service routines.
- Use procedure status values to determine whether procedure calls completed properly.
- Create basic DCL-style commands.
- Create help libraries.

Topics

- Libraries
- The Run-Time Library
- Utility Routines
- System Services
- Procedure Status
- The Command Definition Utility
- Help Libraries

Libraries

Instead of trying to keep track of scattered object modules you can group them together into a library.

- Libraries are files which contain modules and are manipulated and controlled by the Librarian (**$LIBRARY**) utility.
- OpenVMS supports 5 types of libraries:

Library Type	Default Input File Type	Default Library File Type
Object	.OBJ	.OLB
Text	.TXT	.TLB
Help	.HLP	.HLB
Macro	.MAR	.MLB
Shareable Image	.EXE	.OLB

- The Librarian defaults to manipulating object libraries.
- The librarian keeps a history of modifications made to library.
- Modules may be deleted, replaced, inserted, and extracted.

Library Commands

Creating Libraries

```
$ LIBRARY/CREATE/OBJECT  MY_LIB  MOD1,MOD2,MOD3
```

Listing Libraries

```
$ LIBRARY/LIST MY_LIB          !Shows only module names
$ LIBRARY/LIST/NAMES MY_LIB    !Shows global symbols also
```

Extracting Modules

```
$ LIBRARY/EXTRACT=MOD1 /OUTPUT=MOD1 MY_LIB
```

Inserting Modules

```
$ LIBRARY/INSERT MY_LIB MOD4
```

Deleting Modules

```
$ LIBRARY/DELETE=MOD2 MY_LIB
```

Replacing Modules

```
$ LIBRARY/REPLACE MY_LIB MOD3
```

Obtaining Library History

```
$ LIBRARY/LIST/HISTORY MY_LIB
```

Libraries and The Linker

The Linker has global symbol resolution as a responsibility. Global symbols may be resolved from object modules listed on the **$LINK** command line. You may also identify that libraries should be searched using the **/LIBRARY** qualifier.

Linking with libraries example

```
$ LINK/MAP/FULL/EXECUTABLE=PROG -
$_ MAIN,SUB1,LIB1/LIBRARY,LIB2/LIBRARY
```

- The Linker will search a set of libraries automatically. These libraries include:

1. Libraries specified through the logical names:

```
LNK$LIBRARY
LNK$LIBRARY_1
LNK$LIBRARY_2
...
LNK$LIBRARY_999
```

2. The shareable image library **ALPHA$LIBRARY:IMAGELIB.OLB** unless **/NOSYSSHR** or **/NOSYSLIB** specified on link.
3. The object library **ALPHA$LIBRARY:STARLET.OLB** and the shareable image **ALPHA$LIBRARY:SYS$PUBLIC_VECTORS.EXE** unless **/NOSYSSHR** is specified.

Library Example

```
$ type quick_pik.c
void  lotto(int *);
int   get_ball(int, int *);
int   is_member(int,int *);
int   *quick_pick(int *,int);
int   *sort_set(int      *);
void  print(int *);
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
static const int BALLS_PER_DRAW = 6;
static const int FIRST_BALL = 0;
static const int LOW_BALL = 1;

/* Lotto initializer. */
void  lotto(int *ball)
{
     int       i;
     for(i=0;i<BALLS_PER_DRAW;i++)
       ball[i] = 0;
}
/* Function to return a ball value, returns 0 if invalid ball */
int   get_ball(int val, int *ball)
{
     if((val>=0) && (val < BALLS_PER_DRAW))
     {
       return(ball[val]);
     }
     return(0);
}
/* Function to determine whether a value is currently a
   member of the lotto set  Returns Ball value if it is a member
 returns 0 if not found.
*/
```

Library Example (continued)

```
int  is_member(int val,int *ball)
{
    int     i;
    for(i=0;i<BALLS_PER_DRAW;i++)
    {
      if(val == ball[i])
      {
            return(val);
      }
    }
    return(0);
}

/* Generate a lotto quick pick using rand */
const int shift_val = 7;
int  *quick_pick(int *ball,int max_ball)
{
    int     i;
    int     temp;
    srand((int)clock()); /* Set the seed based on the current time*/
    for(i = 0; i < BALLS_PER_DRAW; i++)
    {
      do
      {     /* Generate a relatively random lotto number */
            temp = ((rand()>>shift_val)%max_ball) + 1;
      } while(is_member(temp,ball));  /* If already drawn try again
      /* Set ball */
      ball[i] = temp;
    }
    /* return the quick pick set */
    return(ball);
}
```

Library Example (continued)

```
/* Function to print the current lotto set */
void qp_print(int   *ball)
{
    int     i;
    sort_set(ball);
    printf("Your LUCKY numbers are: ");
    for(i = FIRST_BALL;i<BALLS_PER_DRAW;i++)
      printf("%d ",ball[i]);
    putchar('\n');
}
$
$ cc quick_pik
$ libr/create lotto quick_pik
$
$ library/list lotto
Directory of ALPHA OBJECT library _DKA300:[ALLEN.SHR]LOTTO.OLB;1 on
18-JUL-1996 23:15:47
Creation date:  18-JUL-1996 23:15:21      Creator:  Librarian A09-22
Revision date:  18-JUL-1996 23:15:21      Library format:   3.0
Number of modules:      1                 Max. key length:  128
Other entries:          7                 Preallocated index blocks:    213
Recoverable deleted blocks:      0        Total index blocks used:        2
Max. Number history records:     20       Library history records:        0

QUICK_PIK
$
```

Library Example (continued)

```
$ library/list/names lotto
Directory of ALPHA OBJECT library _DKA300:[ALLEN.SHR]LOTTO.OLB;1 on
18-JUL-1996 23:16:24
Creation date:  18-JUL-1996 23:15:21      Creator:  Librarian A09-22
Revision date:  18-JUL-1996 23:15:21      Library format:   3.0
Number of modules:      1                 Max. key length:  128
Other entries:          7                 Preallocated index blocks:    213
Recoverable deleted blocks:     0         Total index blocks used:        2
Max. Number history records:    20        Library history records:        0

Module QUICK_PIK

GET_BALL
IS_MEMBER
LOTTO
QP_PRINT
QUICK_PICK
SHIFT_VAL
SORT_SET

$
$ del quick_pik.obj;*
$
$ dir quick_pik.obj
%DIRECT-W-NOFILES, no files found
$
```

Library Example (continued)

```
$ libr/extract=quick_pik/output=quick_pik lotto
$ dir quick_pik.obj

Directory _DKA300:[ALLEN.SHR]

QUICK_PIK.OBJ;1

Total of 1 file.
$
$ libr/replace lotto quick_pik
$
$ library/history/list lotto
Directory of ALPHA OBJECT library _DKA300:[ALLEN.SHR]LOTTO.OLB;1 on
18-JUL-1996 23:26:52
Creation date:  18-JUL-1996 23:15:21      Creator:  Librarian A09-22
Revision date:  18-JUL-1996 23:26:48      Library format:   3.0
Number of modules:      1                 Max. key length:  128
Other entries:          7                 Preallocated index blocks:    213
Recoverable deleted blocks:     11        Total index blocks used:        2
Max. Number history records:      20      Library history records:        1

ALLEN        replaced   1 module on 18-JUL-1996 23:26:48
$
$
$
```

Library Example (continued)

```
$ cc gen_qp
$ link gen_qp,lotto/library
$
$ run gen_qp
enter magic number: 22
Your winning numbers are: 2 7 9 19 20 22
$
$ run gen_qp
enter magic number: 42
Your winning numbers are: 15 27 30 31 41 42
$
$ run gen_qp
enter magic number: 13
Your winning numbers are: 8 11 24 30 32 35
$
$ run gen_qp
enter magic number: 22
Your winning numbers are: 3 10 14 24 28 41
$
```

Linker Qualifiers and Libraries

Other qualifiers affecting link symbol resolution

Qualifier	Function
/INCLUDE=(mod1[,mod2,...])	Include only specified modules from library
/LIBRARY/INCLUDE=(mod1[,mod2,...])	Same as include, but also search the library for unresolved symbol references
/SELECTIVE_SEARCH	Reduce symbol table to include only unresolved references. Speeds up link. Reduces MAP file size.
/SYSEXE	Link against SYS$LOADABLE_IMAGES:SYS$BASE_IMAGE.EXE
/USERLIBRARY= ([ALL,GROUP,NONE,PROCESS,SYSTEM])	Identify logical name table search

Text Libraries and C Headers

C header files are implemented using text libraries.

- The following libraries are searched automatically at compile time:

Library	Contents
ALPHA$LIBRARY:DECC$RTLDEF.TLB	C standard and UNIX headers
ALPHA$LIBRARY:STARLETSD.TLB	OpenVMS Run-Time Library function prototypes and structure definitions
ALPHA$LIBRARY:SYS$STARLET_C.TLB	System Service and Record Management Services function prototypes and structure definitions

- The Library **ALPHA$LIBRARY:SYS$LIB_C.TLB** contains function prototypes and structure definitions for internal OpenVMS routines accessed in kernel mode.
- Compiling against a text library:

```
$ CC MY_SYS_PROG+ALPHA$LIBRARY:SYS$LIB_C.TLB/LIBRARY
```

The Run-Time Library

The OpenVMS Run-Time Library (RTL) provides a set of procedures, available to programs regardless of language, which provide system functionality.

- The RTL routines are generally easier to use than system services, although perhaps slower.
- The Run-Time Library procedures are grouped in classes with a unique facility prefix. These classes include:

Prefix	Routine Types
CVT$	Floating point conversion
DNS$	Distributed name services
DTK$	DECTalk
LIB$	General support
OTS$	Object time system
PPL$	Parallel processing
SMG$	Screen management
STR$	String processing

- Sample routines:

```
LIB$CREATE_DIR

LIB$GETJPI

SMG$CREATE_VIRTUAL_DISPLAY

SMG$CREATE_PASTEBOARD
```

Run-Time Library Example

```
$
$ type me.c
/* Example of using run-time library routines.
*/
#include <jpidef.h>
#include <descrip.h>
#include <stdio.h>
#include <ssdef.h>
#include <lib$routines.h>
#include <starlet.h>
#define check(STS) if(!(STS&1)) sys$exit(STS);
#define      USERNAME_SIZE 12
#define      IMAGNAME_SIZE 255

int main(void)
{
     char     user[USERNAME_SIZE];
     char     image[IMAGNAME_SIZE];
     struct   dsc$descriptor_s jpi_desc = {0,DSC$K_DTYPE_T,DSC$K_CLASS_S};
     int      status;
     int      code;
     int      pid=0;

/* Set up descriptor for return string containing user name. */
     jpi_desc.dsc$w_length = USERNAME_SIZE;
     jpi_desc.dsc$a_pointer = user;
```

Run-Time Library Example (continued)

```
/* Define item to get. */
     code = JPI$_USERNAME;
/* Get user name. */
     status = lib$getjpi(&code,0,0,0,&jpi_desc,&jpi_desc.dsc$w_length);
     check(status);
     printf("I am %.*s\n",USERNAME_SIZE,user);
/* Set up descriptor for return string containing image name. */
     jpi_desc.dsc$w_length = IMAGNAME_SIZE;
     jpi_desc.dsc$a_pointer = image;
/* Get image name. */
     code = JPI$_IMAGNAME;
     status = lib$getjpi(&code,0,0,0,&jpi_desc,&jpi_desc.dsc$w_length);
     check(status);
     printf("And I am running ");
/* Display using a descriptor. */
     status = lib$put_output(&jpi_desc);
     check(status);
     return(SS$_NORMAL);
}

$ cc me
$ link me
$ r me
I am ALLEN
And I am running
_DKA300:[ALLEN]ME.EXE;2
$
```

Utility Routines

Several utilities provide a programming interface to their functionality.

Utility Routines

Facility	Utility
ACLEDIT$	**$ EDIT/ACL**
CLI$	DCL command processing
CONV$	**$ CONVERT**
DCX$	Data compression
TPU$	**$ EDIT/TPU**
EDT$	**$ EDIT/EDT**
FDL$	**$ EDIT/FDL**
LBR$	**$ LIBRARY**
LGI$	LOGINOUT
MAIL$	**$ MAIL**
NCS$	National Character Set
PSM$	Print symbiont modification
SMB$	Writing symbionts
SOR$	**$ SORT**

System Services

System services are procedures that provide the closest level a user mode program can get to OpenVMS.

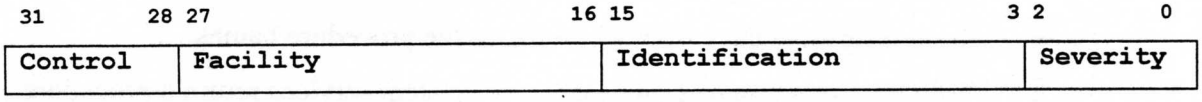
- Most system services run in an elevated access mode (Kernel or Executive) and are entered through a *vector* that issues a **CHMK** or **CHME** PAL code instruction.
- System services, generally, require that all defined arguments must be passed to the procedure.
- System services may be somewhat more difficult to call than other procedures.
- Many system services require additional privilege to call.
- System service vectors are provided in the execlet **SYS$SHARE:SYS$PUBLIC_VECTORS.EXE.**
- System services have a **SYS$** prefix on the procedure names.
- With the exception of **sys$exit,** all system services return a procedure value.

Procedure Status

The OpenVMS Calling Procedure Standard provides a definition for detecting and processing error conditions.

- Success or failure is reported to the calling code in a status longword.
- For most system supplied routines, the calling code must test the condition value to detect errors.
- The condition value is:
  - The function result in high-level languages
  - **R0** in Macro

The condition value



- *Severity* identifies the degree of error

Severity Value	Meaning
0	Warning
1	Success
2	Error
3	Informational
4	Fatal

- *Identification* is used to locate message text.
- *Facility* identifies which component of OpenVMS is returning the condition value.
- *Control* identifies whether message should be displayed by **SYS$EXIT** system service.

Condition Value Processing

When calling system services or RTL routines it is good practice to test for error conditions and process accordingly.

- When testing for an error condition, it is common to test the low bit of the condition value to detect a problem. Low bit set is okay. Low bit clear indicates a problem.
- Sample of testing for an error in C:

```
status = lib$getjpi(&uname_code,0,0,&pid,&jpi_desc,
                    &jpi_desc.dsc$w_length);
if(!(status & 1))   sys$exit(status);
```

- You can also check for a specific error code. System services define their return codes in **ssdef.h**. Other routines will define specific error codes as well.
- Sample of testing for a specific error in C:

```
status = lib$getjpi(&uname_code,0,0,&pid,
                    &jpi_desc,&jpi_desc.dsc$w_length);
if(status == SS$_NOWORLD)
        fprintf(stderr,"You are not worthy!\n");
```

- The header **stsdef.h** provides macros for extracting components of the condition code, such as **$VMS_STATUS_SEVERITY(*code*)** .

Condition Value Processing Example

```
$
$ type status.c
/* Demo of status values.
*/
#include     <stdio.h>
#include     <ssdef.h>
#include     <stsdef.h>
int main(void)
{
     int      severity;
     severity = $VMS_STATUS_SEVERITY(SS$_NOOPER);
     switch(severity)
     {
       case   STS$K_SUCCESS:
              printf("SS$_NOOPER is success\n");
              break;
       case   STS$K_WARNING:
              printf("SS$_NOOPER is warning\n");
              break;
       case   STS$K_INFO:
              printf("SS$_NOOPER is informational\n");
              break;
       case   STS$K_ERROR:
              printf("SS$_NOOPER is error\n");
              break;
       case   STS$K_SEVERE:
              printf("SS$_NOOPER is severe\n");
              break;
     }
```

Condition Value Processing Example (continued)

```
    return(SS$_NOOPER);
}
$
$ cc status
$ link status
$
$ run status
SS$_NOOPER is severe
%SYSTEM-F-NOOPER, operation requires OPER privilege
$
```

Condition Value Handling

When an error is detected, you may want to abort the program.

- You can call the **sys$exit** system service and pass it the condition code. This will abort the program and pass the condition code to DCL if present. DCL will translate and display the message.
- The routine **lib$stop** works the same way as **sys$exit**, except condition handlers (discussed later) will be called prior to termination.
- The routine **lib$signal** works the same way as **lib$stop**, except the program will only terminate if the error is a severe error. If condition handlers do not terminate the program control will be returned to the caller.
- In **main()** you can simply return the condition value.
- DCL provides the status and severity of the last program run in the symbols **$status** and **$severity**, respectively.

LIB$SIGNAL Example

```
$ type status2.c
/* Demo of LIB$SIGNAL with various types of conditions
*/
#include     <stdio.h>
#include     <stdlib.h>
#include     <ssdef.h>
#include     <stsdef.h>
#include     <lib$routines.h>
int  main(void)
{
/* Success code */
     lib$signal(SS$_NORMAL);
/* Informational code */
     lib$signal(SS$_LOGNAME);
/* Warning Code */
     lib$signal(SS$_BADIRECTORY);
/* Error code */
     lib$signal(SS$_PWDWEAK);
     printf("Will get here!!!\n");
/* Severe Error */
     lib$signal(SS$_TOOMANYREDS);
     printf("Won't get here!!!\n");
/* Severe Error */
     lib$signal(SS$_NOOPER);
     return(EXIT_SUCCESS);
}
$
```

LIB$SIGNAL Example (continued)

```
$ cc status2
$ link/notrace status2
$ r status2
%SYSTEM-S-NORMAL, normal successful completion
%SYSTEM-I-LOGNAME, transaction log name is !AD
%SYSTEM-W-BADIRECTORY, bad directory file format
%SYSTEM-E-PWDWEAK, password is too easy to guess; please choose another string
Will get here!!!
%SYSTEM-F-TOOMANYREDS, too many redirects
$
```

LIB$SIGNAL Example (continued)

```
$ link status2
$ r status2
%SYSTEM-S-NORMAL, normal successful completion
%SYSTEM-I-LOGNAME, transaction log name is !AD
%SYSTEM-W-BADIRECTORY, bad directory file format
%TRACE-W-TRACEBACK, symbolic stack dump follows
  image     module     routine              line      rel PC           abs PC
 STATUS2   STATUS2   main                 6435 00000000000000D8 00000000000200D8
 STATUS2   STATUS2   __main                  0 0000000000000054 0000000000020054
                                            0 FFFFFFFF81C62914 FFFFFFFF81C62914
%SYSTEM-E-PWDWEAK, password is too easy to guess; please choose another string
%TRACE-E-TRACEBACK, symbolic stack dump follows
  image     module     routine              line      rel PC           abs PC
 STATUS2   STATUS2   main                 6437 00000000000000EC 00000000000200EC
 STATUS2   STATUS2   __main                  0 0000000000000054 0000000000020054
                                            0 FFFFFFFF81C62914 FFFFFFFF81C62914

Will get here!!!
%SYSTEM-F-TOOMANYREDS, too many redirects
%TRACE-F-TRACEBACK, symbolic stack dump follows
  image     module     routine              line      rel PC           abs PC
 STATUS2   STATUS2   main                 6440 000000000000011C 000000000002011C
 STATUS2   STATUS2   __main                  0 0000000000000054 0000000000020054
                                            0 FFFFFFFF81C62914 FFFFFFFF81C62914

$
```

DCL Commands

DCL (Digital Command Language) acts as the command language interpreter for the system. It is mapped into process P1 space at login.

- DCL processes command verbs using command tables, which are also mapped into process P1 space at login.
- The default command tables are called **SYS$SHARE:DCLTABLES.EXE**.
- There are 2 basic methods for getting a program to be run by typing in a verb other than **$RUN**. They are:

1. Creating a foreign command,
   - using the RTL routine **LIB$GET_FOREIGN** (or using the arguments passed to:

```
int main(int argc, char **args)
```

   - and creating a symbol of the form:

```
$ VERB=="$full-image-file-spec"
```

2. Creating a DCL command by:
   a. Creating a Command Definition Language (**.CLD**) file
   b. Interfacing with DCL using **CLI$** routines
   c. Activating the CLD using the Command Definition Utility **($ SET COMMAND)**

- When using the second method, DCL will parse the command line and provide information about qualifiers and parameters.

The Command Definition Utility

The Command Definition Utility (CDU) uses Command Language Definition (CLD) files to add new *verbs* to the command syntax. CLD syntax is documented in the *OpenVMS Utility Routines Manual*.

- A verb is associated with one or more images. In a simple CLD file, a verb is defined using a **DEFINE VERB** statement and is associated with the following image using the **IMAGE** statement. Multiple images may be grouped under one verb using the **DEFINE SYNTAX** statement.
- Parameters are described using the **PARAMETER** statement. The **PARAMETER** statement allows parameter values to be passed to the program through the routine **CLI$GET_VALUE**. Parameters are labeled using **P1** through **P8** or through the **LABEL** keyword.
- The **PARAMETER** statement allows the use of the following clauses:
  - DEFAULT
  - LABEL=*name*
  - PROMPT
  - VALUE([[NO]CONCATENATE,DEFAULT=*string*,LIST, REQUIRED, TYPE=*type*])
- Qualifiers may also be defined using the **QUALIFIER** statement and detected using the routine **CLI$PRESENT**. Their value (if any) may be obtained from **CLI$GET_VALUE**.

The Command Definition Utility (continued)

- **QUALIFIER** clauses:
  - BATCH
  - DEFAULT
  - LABEL=*name*
  - [NON]NEGATABLE
  - PLACEMENT=([GLOBAL,LOCAL,POSITIONAL])
  - SYNTAX=*name* (see DEFINE SYNTAX)
  - VALUE(DEFAULT=*string*,LIST,REQUIRED,TYPE=*type*)
- Types may be defined so that DCL will validate the input value on the command line.
- Valid **TYPE**s:
  - $ACL
  - $DATETIME
  - $DELTATIME
  - $EXPRESSION
  - $FILE
  - $NUMBER
  - $PARENTHESIZED_VALUE
  - $QUOTED_STRING
  - $REST_OF_LINE

Note

CDU supports linking against an object CLD and using **CLI$DCL_PARSE** and **CLI$DISPATCH** to process the command through internal routines. This method is not discussed in this course, but is used by commands like the **INSTALL** utility which process multiple commands.

CDU Routines

Routines supporting the CDU interface

- To determine presence of a qualifier or parameter:

```
CLI$PRESENT(entity_descriptor)
```

Returns:

```
CLI$_PRESENT, CLI$_ABSENT, CLI$_NEGATED, CLI$_DEFAULTED,
CLI$_LOCPRES, CLI$_LOCNEG, CLI$_INVREQTYP.
```

- To obtain a qualifier or parameter value:

```
CLI$GET_VALUE(entity_descriptor,ret_descriptor [,ret_len])
```

Returns:

```
CLI$_COMMA, CLI$_CONCAT, SS$_NORMAL, CLI$_ABSENT,
CLI$_INVREQTYP.
```

Using CDU

- To enable a process specific command:

```
$SET COMMAND cld_file    !Defaults to .CLD file type
```

- To delete a verb:

```
$SET COMMAND/DELETE=verb
```

- To create a complete set of tables:

```
$SET COMMAND cld_file -
$_ /TABLE=SYS$SHARE:DCLTABLES.EXE/OUTPUT=new_tables
```

- To set up a table to be mapped to a user at login:

1. Copy your tables to **SYS$SHARE:**
2. Make sure READ and EXECUTE file protections are on for WORLD
3. $ INSTALL ADD SYS$SHARE:*your_tables*/SHARE
4. Modify the account, using **MODIFY *user* /CLITABLES=*your_tables*** using the **AUTHORIZE** utility.

Note
Using **SYS$SHARE:DCLTABLES.EXE** as your output tables will change the system-wide tables!

Simple CDU Example

```
$ type simple.cld
define verb simple
     image  ALLEN$DKA200:[BAE]simple.exe
     parameter P1
     qualifier do_it, negatable
$
$ simple
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \SIMPLE\
$ set command simple
$
$ simple
%DCL-W-ACTIMAGE, error activating image ALLEN$DKA200:[BAE]SIMPLE.EXE
-CLI-E-IMAGEFNF, image file not found ALLEN$DKA200:[BAE]SIMPLE.EXE;
$
$ type simple.c
/* Sample program to illustrate simple DCL command
   line processing.

*/
#include <cli$routines.h>
#include <descrip.h>
#include <ssdef.h>
#include <stdio.h>
#define CLI_MAX_LINE 255
```

Simple CDU Example (continued)

```
int main(void)
{
    $DESCRIPTOR(p1_param,"P1");
    $DESCRIPTOR(do_it_qual,"DO_IT");
    static  char  param_str[CLI_MAX_LINE+sizeof('\0')];
    struct  dsc$descriptor_s param_value = {CLI_MAX_LINE,
                DSC$K_DTYPE_T,DSC$K_CLASS_S,
                param_str};

/* Get the parameter, if entered and echo it back. */
    if(cli$get_value(&p1_param,&param_value,
            &param_value.dsc$w_length) & SS$_NORMAL)
    {
      param_str[param_value.dsc$w_length] = '\0';
      printf("Paramater == %s\n",param_str);
    }
    else
    {
      puts("No parameter entered!");
    }
```

Simple CDU Example (continued)

```
/* Check for entry of qualifier. */
     if(cli$present(&do_it_qual)&SS$_NORMAL)
     {
       puts("Okay, okay, already.  I will do it!");
     }
     else
     {
       puts("thanks");
     }
     return(SS$_NORMAL);
}
$
$ cc simple
$ link simple
$
$ simple
No parameter entered!
thanks
$
$ simple task/do_it
Paramater == TASK
Okay, okay, already.  I will do it!
$
$ simple "Just ask"/nodo_it
Paramater == Just ask
thanks
$
```

CDU Example

```
$
$ type ytd.cld
define verb   ytd
    image    dka300:[ALLEN]ytd
    parameter      p1,value(type=$datetime,default=NOW)
    qualifier      month
    qualifier      day,default,negatable
    qualifier      hour
    qualifier      minute
    qualifier      second
    qualifier      symbol
    qualifier      global
    disallow       global and not symbol
    qualifier      display,default,negatable
$
$ type ytd.c
/*
    Example of using cli routines to process a new DCL
    command.

    Command format:

      year [time]

      Qualifiers:

      /MONTH       month of year
      /[NO]DAY     day of year
```

CDU Example (continued)

```
        /HOUR           hour of year
        /MINUTE         minute of year
        /SECOND         second of year
        /[NO]DISPLAY            display on SYS$OUTPUT
        /SYMBOL         create symbols based on time qualifiers
        /GLOBAL         make the symbols global


*/
/* Include headers. */
#include <stdlib.h>
#include <stdio.h>
#include <lib$routines.h>
#include <ots$routines.h>
#include <cli$routines.h>
#include <starlet.h>
#include <libdtdef.h>
#include <descrip.h>
#include <ssdef.h>
#include <ctype.h>
#include <clidef.h>
#include <libclidef.h>
#include <string.h>
#include <assert.h>
```

CDU Example (continued)

```
/* Define constants and macros. */
#define TIMESTR_MAX 23
#define TIMEINT_MAX 2

#define YES 1
#define NO 0
#define sys_stat(STS)   (((STS)&1))
#define MAX_INT_STR 10
#define NO_SYMBOL_QUALIFIER 0
void process_qualifiers(struct dsc$descriptor_s *qid_d,
            const long *cnv_type,int  *sym_flag,
            char   *header,int *time,int disp_flag);
/*********************************
    Start of program
*********************************/
int  main(void)
{
/*** Data declaration/definition start ***/

    char     time_s[TIMESTR_MAX+1];
    struct   dsc$descriptor time_d= {sizeof(time_s),DSC$K_DTYPE_T,
                        DSC$K_CLASS_S};
    int      time[TIMEINT_MAX];
    int      stat;
    $DESCRIPTOR(day_q,"DAY");
    $DESCRIPTOR(month_q,"MONTH");
    $DESCRIPTOR(hour_q,"HOUR");
```

CDU Example (continued)

```
    $DESCRIPTOR(min_q,"MINUTE");
    $DESCRIPTOR(sec_q,"SECOND");
    $DESCRIPTOR(symbol_q,"SYMBOL");
    $DESCRIPTOR(display_q,"DISPLAY");
    $DESCRIPTOR(global_q,"GLOBAL");
    $DESCRIPTOR(param,"P1");
    const    long   MOY = LIB$K_MONTH_OF_YEAR;
    const    long   DOY = LIB$K_DAY_OF_YEAR;
    const    long   HOY = LIB$K_HOUR_OF_YEAR;
    const    long   moy = LIB$K_MINUTE_OF_YEAR;
    const    long   SOY = LIB$K_SECOND_OF_YEAR;
    int      sym_flag;
    int      display = NO;
/*** Data declaration/definition END ***/
/* Make sure no conflict exists in the following symbols. */
assert(NO_SYMBOL_QUALIFIER != LIB$K_CLI_LOCAL_SYM);
assert(NO_SYMBOL_QUALIFIER != LIB$K_CLI_GLOBAL_SYM);

/* Set up time string descriptor. */
    time_d.dsc$a_pointer = time_s;
/* Get time from command line, if NOW use current time. */
    stat = cli$get_value(&param,&time_d,&time_d.dsc$w_length);
    sys_stat(stat);
```

CDU Example (continued)

```
/* Make time a C string. */
    time_s[time_d.dsc$w_length] = '\0';

/* Convert time to binary internal representation. */
    if(strcmp("NOW",time_s) == 0)
    {
      stat = sys$gettim(time);
    }
    else
    {
      stat = lib$convert_date_string(&time_d,time);
    }
    sys_stat(stat);
/* Set qualifier based flags for /DISPLAY, /SYMBOL, and /GLOBAL. */
    if(sys_stat(cli$present(&display_q)))
    {
      display = YES;
      if(sys_stat(cli$present(&global_q)))
      {
            sym_flag = LIB$K_CLI_GLOBAL_SYM;
      }
    }
    if(sys_stat(cli$present(&symbol_q)))
    {
      sym_flag = LIB$K_CLI_LOCAL_SYM;
      if(sys_stat(cli$present(&global_q)))
      {
            sym_flag = LIB$K_CLI_GLOBAL_SYM;
      }
    }
```

CDU Example (continued)

```
    else
    {
      sym_flag = NO_SYMBOL_QUALIFIER;
    }
/* Process the time based qualifiers and either display or set up
   appropriate symbols.
*/
    /* /MONTH */
    process_qualifiers(&month_q,&MOY,&sym_flag,
            "Month of year:",time,display);
    /* /DAY */
    process_qualifiers(&day_q,&DOY,&sym_flag,
            "Day of year:",time,display);
    /* /HOUR */
    process_qualifiers(&hour_q,&HOY,&sym_flag,
            "Hour of year:",time,display);
    /* /MINUTE */
    process_qualifiers(&min_q,&moy,&sym_flag,
            "Minute of year:",time,display);
    /* /SECOND */
    process_qualifiers(&sec_q,&SOY,&sym_flag,
            "Second of year:",time,display);

}
```

CDU Example (continued)

```
/***********************************************
    Function to process command qualifiers

    Inputs:
    1)      Time-based Qualifier descriptor address
    2)      long conversion type address (For ots$cvt_l_ti)
    3)      Symbol type flag address (For CLI$SET_SYMBOL)
    4)      Header string address for display
    5)      Internal time address
    6)      /[NO]DISPLAY flag

    Outputs:
    None
    Side effects:
    May terminate command if error detected.
***********************************************/
void process_qualifiers(struct dsc$descriptor_s *qid_d,
            const long *cnv_type,int  *sym_flag,
            char   *header,int *time,int disp_flag)
{
    int     status;
    long    num_since_new_yr;
    char    equiv_s[MAX_INT_STR];

    struct  dsc$descriptor equiv_d= {sizeof(equiv_s),DSC$K_DTYPE_T,
                        DSC$K_CLASS_S};
```

CDU EXAMPLE (continued)

```
    equiv_d.dsc$a_pointer = equiv_s;
/* determine if qualifier is present, if not just return. */
    if(sys_stat(cli$present(qid_d)))
    {
    /* get number of time units since midnignt Jan 1 */
      status = lib$cvt_from_internal_time(cnv_type,
                  &num_since_new_yr,time);
      sys_stat(status);
      if(disp_flag) printf("%s %d\n",header,num_since_new_yr);
    /* if symbol to be set create it. */
      if(*sym_flag != NO_SYMBOL_QUALIFIER)
      {
      /* Generate a string from the number of units. */
            status = ots$cvt_l_ti(&num_since_new_yr,&equiv_d);
            sys_stat(status);
      /* Skip leading whitespace from conversion. */
            while(isspace(*equiv_d.dsc$a_pointer))
            {
                  equiv_d.dsc$a_pointer++;
                  equiv_d.dsc$w_length--;
            }

      /* create the symbol. */
            status = lib$set_symbol(qid_d,&equiv_d,
                                    sym_flag);
            sys_stat(status);

      }
    }
}
$
```

CDU EXAMPLE (continued)

```
$ cc ytd
$ link/notrace ytd
$
$ set command ytd
$
$ ytd
Day of year: 202
$
$ ytd/nodisplay/hour/symbol
$ show symbol/all
  DAY = "202"
  HOUR = "4825"
$
$ ytd/nodisplay/hour/symbol 25-DEC
$ show symbol/all
  DAY = "360"
  HOUR = "8616"
$
$ ytd tomorrow
Day of year: 203
$
```

CDU Example (continued)

```
$ ytd yesterday
Day of year: 201
$ ytd garbage
%DCL-W-IVATIME, invalid absolute time - use DD-MMM-YYYY:HH:MM:SS.CC format
 \GARBAGE\
$ ytd/symbol/global
Day of year: 202
$ show sym/glob/all
  $RESTART == "FALSE"
  $SEVERITY == "1"
  $STATUS == "%X00000001"
  DAY == "202"
  ED*IT == "edit/section=sys$login:mysec.eve"
$ ytd/global/hour
%DCL-W-CONFLICT, illegal combination of command elements - check documentation
 \GLOBAL\
$ set comm/tables=sys$share:dcltables
$ ytd
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \YTD\
$ dir sys$share:dcltables

Directory SYS$COMMON:[SYSLIB]

DCLTABLES.EXE;109   DCLTABLES.TEMPLATE;1

Total of 2 files.
$
```

CDU Example (continued)

```
$ set command/tables=sys$share:dcltables/out=sys$share:dcltables ytd
$
$ dir sys$share:dcltables

Directory SYS$SYSROOT:[SYSLIB]

DCLTABLES.EXE;1

Total of 1 file.

Directory SYS$COMMON:[SYSLIB]

DCLTABLES.EXE;109   DCLTABLES.TEMPLATE;1

Total of 2 files.

Grand total of 2 directories, 3 files.
$
$ ytd
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \YTD\
$
$ set comm/tables=sys$share:dcltables
$
$ ytd
Day of year: 202
$
```

CDU Example (continued)

```
$ delete sys$specific:[syslib]dcltables.exe;
$
$ dir sys$share:dcltables

Directory SYS$COMMON:[SYSLIB]

DCLTABLES.EXE;109   DCLTABLES.TEMPLATE;1

Total of 2 files.
$
$
```

HELP Libraries

As you write new utilities or programs you may want to provide help using the standard **$ HELP** utility.

- The **$ HELP** utility works with help libraries. These libraries are created and manipulated in much the same way as object libraries, using the **$ LIBRARY** (librarian) utility with the **/HELP** qualifier.
- Help file modules work off of the first column. The first column of text can contain the following characters:

  1 Identifies primary help keyword

  2-9 Identifies level of sub-help keyword

  / Identifies qualifier help keyword
- It is assumed that help files will have a **.HLP** file type and help libraries will have a **.HLB** file type.
- The default help library searched by the **$ HELP** utility is **SYS$HELP:HELPLIB.HLB.**
- You may specify that an alternate library should be searched using: **$HELP/LIBRARY=*help-library-filespec*.** The default directory searched is **SYS$HELP:**.
- You may also have your library included in the standard help display using the logical names: **HLP$LIBRARY, HLP$LIBRARY_1, HLP$LIBRARY_2, ... HLP$LIBRARY_999.**
- You can deliver help from within a program by using the utility routines: **LBR$INI_CONTROL, LBR$OPEN,** and **LBR$GET_HELP** or **LBR$OUTPUT_HELP.**

HELP Library Example

```
$
$ type ytd.hlp
1 YTD

     The YTD command will display the unit of
     year since 1-JAN-YYYY 00:00:00.00.  Units
     may be specified in: days, months, hours, minutes,
     and seconds (default is days).  You may also
     generate a symbol containing the unit.

     Format:

     $ YTD [absolute-time]

2 Parameter

     absolute-time

     Specify the date and time you want to use for
     defining the unit of year.  Defaults to NOW.

3 Example

     $ YTD 25-DEC-1996
     Day of year: 360
     $
```

HELP Library Example (continued)

```
2 Qualifiers

    Help is available on the following qualifiers:

    /[NO]DAY /MONTH       /HOUR
    /MINUTE       /SECOND             /SYMBOL
    /[NO]DISPLAY  /GLOBAL

/[NO]DAY
    /NODAY turns off default day of year display.
    When used with /SYMBOL qualifier, creates a
    symbol called DAY containing day of year.
/MONTH
    Causes month of year to be displayed.
    When used with /SYMBOL qualifier, creates a
    symbol called MONTH containing month of year.
/HOUR
    Causes hour of year to be displayed.
    When used with /SYMBOL qualifier, creates a
    symbol called HOUR containing hour of year.
/MINUTE
    Causes minute of year to be displayed.
    When used with /SYMBOL qualifier, creates a
    symbol called MINUTE containing minute of year.
```

HELP Library Example (continued)

```
/SECOND
     Causes second of year to be displayed.
     When used with /SYMBOL qualifier, creates a
     symbol called SECOND containing second of year.
/SYMBOL
     Causes a symbol or symbols for each unit qualifier
     used.  Symbol name matches unit qualifier name.
/GLOBAL
     Makes symbol global.  Must be used with /SYMBOL
     qualifier.


$
$
$ library/create/help year ytd
```

HELP Library Example (continued)

```
$ libr/list/help year
Directory of HELP library _DKA300:[ALLEN]YEAR.HLB;1 on 20-JUL-1996 01:44:03
Creation date:  20-JUL-1996 01:43:38      Creator:  Librarian A09-22
Revision date:  20-JUL-1996 01:43:38      Library format:   3.0
Number of modules:      1                 Max. key length:  15
Other entries:          0                 Preallocated index blocks:      5
Recoverable deleted blocks:      0        Total index blocks used:        1
Max. Number history records:     20       Library history records:        0

YTD
$
$ help/library=dka300:[Allen]year

  Sorry, no documentation on HELP

  Additional information available:

  YTD

Topic? ytd

YTD

     The YTD command will display the unit of
```

HELP Library Example (continued)

```
year since 1-JAN-YYYY 00:00:00.00.  Units
may be specified in: days, months, hours, minutes,
and seconds (default is days).  You may also
generate a symbol containing the unit.

Format:

$ YTD [absolute-time]



Additional information available:

Parameter  Qualifiers
/[NO]DAY   /MONTH     /HOUR      /MINUTE    /SECOND    /SYMBOL    /GLOBAL
```

HELP Library Example (continued)

```
YTD Subtopic? param

YTD

  Parameter

    absolute-time

    Specify the date and time you want to use for
    defining the unit of year.  Defaults to NOW.


    Additional information available:

    Example

YTD Parameter Subtopic? exampl

YTD

  Parameter

    Example

      $ YTD 25-DEC-1996
      Day of year: 360
      $

YTD Parameter Subtopic?
```

HELP Library Example (continued)

```
YTD Subtopic? qual

YTD

  Qualifiers


    Help is available on the following qualifiers:

    /[NO]DAY /MONTH        /HOUR
    /MINUTE          /SECOND                /SYMBOL
    /[NO]DISPLAY     /GLOBAL

    /[NO]DAY
    /NODAY turns off default day of year display.
    When used with /SYMBOL qualifier, creates a
    symbol called DAY containing day of year.
    /MONTH
    Causes month of year to be displayed.
    When used with /SYMBOL qualifier, creates a
    symbol called MONTH containing month of year.
    /HOUR
    Causes hour of year to be displayed.
    When used with /SYMBOL qualifier, creates a
    symbol called HOUR containing hour of year.
    /MINUTE
    Causes minute of year to be displayed.
```

HELP Library Example (continued)

```
     When used with /SYMBOL qualifier, creates a
     symbol called MINUTE containing minute of year.
     /SECOND
     Causes second of year to be displayed.
     When used with /SYMBOL qualifier, creates a
     symbol called SECOND containing second of year.
     /SYMBOL
     Causes a symbol or symbols for each unit qualifier
     used.  Symbol name matches unit qualifier name.
     /GLOBAL
     Makes symbol global.  Must be used with /SYMBOL
     qualifier.
YTD Subtopic? /global

YTD

  /GLOBAL
     Makes symbol global.  Must be used with /SYMBOL
     qualifier.



YTD Subtopic?
Topic?
```

HELP Library Example (continued)

```
$ define hlp$library dka300:[Allen]year.hlb
$ help

HELP

   The HELP command invokes the HELP Facility to display
   information about a command or topic. In response to the "Topic?"
   prompt, you can:

      o Type the name of the command or topic for which you need help.

      o Type INSTRUCTIONS for more detailed instructions on how to use
        HELP.

      o Type HINTS if you are not sure of the name of the command or
        topic for which you need help.

      o Type HELP/MESSAGE for help with the HELP/MESSAGE utility.

      o Type a question mark (?) to redisplay the most recently requested
        text.

      o Press the Return key one or more times to exit from HELP.

   You can abbreviate any topic name, although ambiguous abbreviations
   result in all matches being displayed.
```

HELP Library Example (continued)

```
  Additional information available:

  :=          =           @           ACCOUNTING  ALLOCATE    ANALYZE     APPEND
  ASSIGN      ATTACH      AUTHORIZE   AUTOGEN     BACKUP      CALL        CANCEL
  CC          CLOSE       CONNECT     CONTINUE    CONVERT     COPY        CREATE
...
  SYSGEN      SYSMAN      System_Files            System_Services         TFF
  TYPE        UIL         UNLOCK      V70_Features            VEST        VIEW
  WAIT        WRITE
Additional help libraries available (type @name for topics):

  YEAR

Topic? @year

  Sorry, no documentation on HELP


  Additional information available:

  YTD

  Additional help libraries available (type @name for topics):

  YEAR

@YEAR Topic? ytd
```

HELP Library Example (continued)

```
YTD


     The YTD command will display the unit of
     year since 1-JAN-YYYY 00:00:00.00.  Units
     may be specified in: days, months, hours, minutes,
     and seconds (default is days).  You may also
     generate a symbol containing the unit.

     Format:

     $ YTD [absolute-time]



  Additional information available:

  Parameter  Qualifiers
  /[NO]DAY   /MONTH     /HOUR      /MINUTE    /SECOND    /SYMBOL    /GLOBAL
@YEAR YTD Subtopic?
@YEAR Topic?
Topic?
$
$ deas hlp$library
$
```

HELP Library Example (continued)

```
$
$ library/help sys$help:helplib ytd
%LIBRAR-F-OPENIN, error opening SYS$COMMON:[SYSHLP]HELPLIB.HLB;3 as input
-RMS-E-PRV, insufficient privilege or file protection violation
$
$ set proc/priv=sysprv
$ library/help sys$help:helplib ytd
$
$ help

HELP

   The  HELP  command  invokes  the  HELP  Facility  to  display
   information about a command or topic.  In response to the "Topic?"
   prompt, you can:

      o Type  the  name of the command  or topic for which you need help.
...
  STOP        SUBMIT      SUBROUTINE Symbol_Assign          SYNCHRONIZE
  SYSGEN      SYSMAN      System_Files           System_Services       TFF
  TYPE        UIL         UNLOCK      V70_Features          VEST       VIEW
  WAIT        WRITE       YTD

Topic?
```

HELP Library Example (continued)

```
$ help ytd

YTD

     The YTD command will display the unit of
     year since 1-JAN-YYYY 00:00:00.00.  Units
     may be specified in: days, months, hours, minutes,
     and seconds (default is days).  You may also
     generate a symbol containing the unit.

     Format:

     $ YTD [absolute-time]

   Additional information available:

   Parameter  Qualifiers
   /[NO]DAY   /MONTH     /HOUR      /MINUTE    /SECOND    /SYMBOL    /GLOBAL

YTD Subtopic?
Topic?
$
$ library/help/delete=ytd sys$help:helplib
$
$ set comm/tables=sys$share:dcltables
$ ytd
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \YTD\
$
```

Getting Help from a Program Example

```
$
$ type get_help.hlp
1    HELP
1    INSTRUCTOR

     Dylan is the instructor.

2    NAME

     Dylan Allen

2    AGE

     29 (of course!)

$
$ library/create/help my_help get_help
$
$ type get_help.c
/* Sample of getting help displayed from within a
   program.

*/

#include <lbr$routines.h>
#include <lib$routines.h>
```

Getting Help From a Program Example (continued)

```
#include <descrip.h>
#include <ssdef.h>
#include <stdio.h>
#include <starlet.h>
int main(void)
{
    $DESCRIPTOR(helplib,"ALLEN$DKA200:[BAE]MY_HELP.HLB");
       int  status;

/* Get help, let lib$put_output and lib$get_input do the I/O. */
    if(!((status=lbr$output_help(lib$put_output,0,0,&helplib,0,
            lib$get_input))&SS$_NORMAL))
    {
      fprintf(stderr,"No help for you\n");
      sys$exit(status);
    }
    return(status);
}

$
$ cc get_help
$ link get_help
$
$
$ run get_help
```

Getting Help From a Program Example (continued)

```
Information available:

  HELP        INSTRUCTOR

Topic? ins
INSTRUCTOR


    Dylan is the instructor.



  Additional information available:

  NAME        AGE

INSTRUCTOR Subtopic? name

INSTRUCTOR

  NAME


    Dylan Allen
```

Getting Help From a Program Example (continued)

```
INSTRUCTOR Subtopic? age

INSTRUCTOR

  AGE


     29 (of course!)


INSTRUCTOR Subtopic?
Topic?
$
```