DPL - COMMERCIAL JOB CONTROL LANGUAGE

B. A. Harper
Computerquest, Brisbane.

T. H. Croll
Digital Products Pty. Ltd., Brisbane.

ABSTRACT

DPL is a general purpose interpreted language designed to reduce the system support needed in maintaining high level user interfaces to complex collections of software. Originally devised especially for TSX-Plus, DPL also operates equally well under RT-11 and VMS. The language has been designed to fill the gap between basic indirect command files and the more powerful but essentially rigid structures of IND for commercial applications.

INTRODUCTION

DPL (Digital Products Language) is a simple command language having a small number of directives but with a powerful underlying design structure which suits itself to the special needs of commercial applica tions. Novice users can quickly gain confidence in setting up otherwise complex command procedures.

The language is "IND-like" in some of its operations but concentrates on ease-of-use, flexability of input/output, terminal independent screen control and powerful user-interrupt handling procedures. DPL, being a general purpose interpreted language, is not subject to the constraints of structure imposed by more conventional menu control systems such as SPECS (Ref 1), as developed previously by the senior author. The DPL language is undergoing continued development and many further features are planned. Source language is the powerful FORth generation language TRAN.

DPL DESIGN FEATURES

General Operation

DPL provides a CSI interface through which file and control information may be passed, e.g.

```
     .RUN DPL {outfilespec/switch=}infilespec{/switch}
  or
     .RUN DPL
     *{outfilespec/switch=}infilespec{/switch}
```

It is assumed DPL.SAV resides on logical device DPL:. Default operation is selected by specifying /D as the command input. This will select the default input file DPL:MASTER.MNU for processing.

Under TSX-Plus a user-defined command would be typically, e.g.

```
     DPL :== RUN/S DPL:DPL /D
```

where /S signifies use of single character activation mode. Use of /H at the CSI prompt provides help on further DPL processing options which are discussed in later sections.

An initial first pass of the input file provides for stripping of comments, decoding of command lines, checking of directives, label and data table uniqueness and sets up argument addresses, storing each user command in a compressed format to preserve memory. No syntax checking is performed at this stage. If the input filetype is ".MNC" it is assumed to be a pre compiled source file in binary format and DPL bypasses the initial phase.

A number of internally generated (permanent) symbol values are then initialised before execution of the command file. Symbol and data table space is dynamically allocated and managed by squeezing if necessary in response to the storage needs of the command file.

Execution flow control may cause transfer to any number of other files containing further DPL instructions. User symbols are normally purged during such transfers of control but may optionally be saved if required to allow sharing of values. The chain area is used for sending commands directly to the monitor and, while under monitor control, the present status of the DPL file is saved on a scratch file to allow later recovery to the next DPL command in sequence. A maximum of 127 command lines is presently allowed in any single command file.

Directives

Table 1 summarises the DPL directives currently available. Simplicity in source file decoding is aided by allowing only one directive per command line. In practice this is not a handicap to efficient programming because many directives contain implicit "if-then-else" logic.

TABLE 1
DPL DIRECTIVE SUMMARY

Name	Usage
$ACCEPT	symbol data
$BRANCH_ON_INPUT	symbol data
$CLEAR	screen
$DCL	monitor command
$DELETE	symbol
$DISPLAY	text
$DEF_TABLE	pure data
$END_TABLE	pure data
$EXIT	exit
$GOSUB	subroutine
$GOTO	label
$IF	relational tests
$INPUT	symbol
$LOOKUP	symbol in table
$RETURN	from gosub
$SAVE	symbols on exit
$SET	symbol value
$TESTFILE	if exist
$TESTLABEL	if exist
$WAIT	for seconds

Symbols

Symbols are either permanent or user-defined. All symbols contain variable length strings up to a maximum of 80 characters each. Symbols may be used to build string expressions via concatenation (+) with other symbols and literal text enclosed in double quotation marks " ".

Symbol Substitution

This feature is always enabled and provides a single level of substitution only (by intent). No special parenthesis is needed to invoke substitution - it occurs as a normal consequence of the particular command syntax.

Labels

Labels may be specified for any or all command lines. Symbols may contain label specifications.

Image Text

Lines devoid of any type of label or DPL directive are shown directly on the screen by default. Typically this feature is used for whole screen displays, help information etc.

Generalised GOTO

DPL allows branching to labels anywhere within a file and, if a label cannot be satisfied internally, it is automatically assumed to be a file specification and control is transfered accordingly. This allows easy segmentation of large procedures into separate sub-files with the one level of GOTO logic.

Terminal Independence

A powerful terminal independent interface provides support for VT52, VT100, ADM and HAZELTINE emulation automatically under TSX-Plus and VMS. With RT-11 a CSI switch can be used to specify the terminal type. The terminal interface allows X,Y cursor positioning, arrow key controls, clear screen and keypad activation. as well as specifying character and line deletion keys. Special activation sequences defining logical commands such as <HELP>, <ABORT>, <ENTER> etc are also available. All key definitions are soft and easily changed on order to provide compatibility with existing site standards such as KED or HARTLEY HAPAS. Table 2 shows a typical soft interface definition for a VT100 terminal. The specified <system-abort> character acts in a special "breakthrough" mode regardless of any preceding key sequences, to allow a hard abort function.

Single Line Editor

An internal single line editor operates in conjunction with the terminal independent interface and allows deletion and overtype editing within a protected field, the length of which is set by the directive calling it.

If-Then-Else Logic

Logical branches are based on an implied "else" logic with "then" command continuation on the immediately following line. This allows more concise expression of logical flow in a block-like manner.

Compiled Source Files

DPL input files may be either ASCII source or "compiled" binary. Compiling, a run-time option, reduces initial decoding overheads. DPL also has the ability to restore a binary file to its ASCII source if necessary and password decode protection can be included to protect against unwelcome hackers.

Subroutines

In-file subroutines are permitted up to ten levels of nesting. "Subroutine" files are possible via the use of DPL maintained permanent symbols which keep track of the last used filename.

Lookup Tables

Pure data sections may be defined in terms of lookup tables to enable easy checking of user inputs against allowable responses, options or filenames.

TABLE 2
EXAMPLE TERMINAL INTERFACE FUNCTIONS

Function	VT100 default
<system-abort>	^C
<left-arrow>	$[D
<right-arrow>	$[C
<delete-char-left>	"177
<delete-at-cursor>	<keypad> ,
<delete-line>	PF4
<backspace>	"10
<return-string>	RETURN
<up-arrow>	$[A
<down-arrow>	$[B
<help>	PF2
<soft-abort>	PF1
<alternate-enter>	ENTER
<move cursor>	$[y;xH
<set-keypad>	$=
<unset-keypad>	$>
<clear-to-EOS>	$[Jn

Verification of Labels and Filenames

Labels within the current file and filenames may be checked for existence prior to acceptance of user requests.

Automatic BRANCH ON INPUT

A single command can be used to prompt for a single character user input and automatically branch to the corresponding single character label in the current file. If no label is available the user input is rejected. This feature is ideal for menu style selections.

Access Protection and Security

Through the use of permanent symbols, DPL programs have access to project-programmer numbers and usernames under TSX-Plus and VMS which can be used to limit access to the system. Control-C trapping also can be used to prevent return to the monitor. Passwords can be protected by specifying no-echo of user input.

Interrupt Service Labels

During input mode the terminal interface will react immediately to certain pre-specified key sequences and attempt to transfer control to specified labels in the current file. This powerful feature enables on-line HELP, error trapping and numerous other services.

DCL Commands

Complex series of DCL commands may be constructed from symbol data and text strings and passed back to the monitor for execution. An automatic return to DPL is included in the DCL sequence and execution continues at the command line following the last DCL command specified.

General File Structure

DPL makes no assumptions about the relationships between the various files which may make up a particular DPL system. However, as an aid to file management a number of permanent symbols are maintained by DPL. These include the filename of the "master" or original file used to invoke the system, the current filename, the last-used filename and the next-to-use filename after a DCL command. Also, a symbol is available to store the "tree" or logical connecting file linking the current file into some pre-determined structure.

A TYPICAL DPL APPLICATION

Figure 1 shows a possible KED "word processing" system WP.MNU written in DPL which illustrates the use of many of the language directives. Figure 2 is an associated DPL file HELP.MNU which is called from WP.MNU to provide on-line help, but which could also be called as a "subroutine" file from other DPL files within an integrated system.

The major features of WP.MNU are as follows:

- the interrupt label <SOFABO> is first set equal to label "TOP" to allow a soft-abort during data entry.

- the screen is cleared and date, time and username displayed.

FIGURE 1
Example Word Processing System

```
! WP.MNU - DECUS Australia Symposium 1985

$SET <sofabo> "top"
:TOP:$CLEAR
$DISPLAY 1 3 <date>" "<time>
$DISPLAY 60 3 <user>

        W O R D   P R O C E S S I N G

              1  Edit Existing File

              2  Create New File

              3  File Directory

              4  Print a File

              E  Exit

$DISPLAY 8 23 "Press PF1 to ABORT, PF2 for HELP"
$DISPLAY 13 18 "Option ? : "
$BRANCH 24 18 opt

! Edit existing file
:1:$SET switch " "
$GOSUB "infile"
$TESTFILE file nofile
:1A:$DCL "EDIT "file+switch
$GOTO "top"

! Create new file
:2:$SET switch "/CREATE"
$GOSUB "infile"
$GOTO "1a"

! Directory of files
:3:$CLEAR
$DCL "DIR/ORDER:NAME"
$DCL "DIR/FREE"
$GOTO "top"

! Print a file
:4:$GOSUB "infile"
$TESTFILE file nofile
$DISPLAY 13 20 "Which printer ? : "
:4A:$ACCEPT 31 20 2 "LS" lpt
$LOOKUP lpt plist nolpt
    $DISPLAY 13 21 file" queued to printer "lpt
    $DCL "PRINT/NAME:"lpt": "file
    $GOTO "top"

! Subroutine for filename prompt
:INFILE:$DISPLAY 13 19 "Filename ? :"
$ACCEPT 24 19 10 "defaul.txt" file
$RETURN

! Error handling
:NOFILE:$DISPLAY 24 20 "File "file" does not exist.. "
$WAIT "3"
$CLEAR 1 20
$GOTO opt

:NOLPT:$DISPLAY 13 21 lpt" is not a valid printer..."
$WAIT "3"
$CLEAR 1 21
$GOTO "4a"
```

```
! Valid printer table
$DEFTABLE plist
LS
LP
LQ
$ENDTABLE

! HELP interrupt handling
:HELP:$DISPLAY 13 21 "Press <space> to return to menu"
$DISPLAY 13 22 "or enter menu option for help :"
$INPUT 44 22 hlp
$IF hlp ne " " "top"
    $TESTLABEL hlp "nohlp"
    $SAVE
    $GOTO "HELP.MNU"
:NOHLP:$DISPLAY 13 23 hlp" is not a valid option.  "
$WAIT "3"

! Error traps
:ERROR:
:C:$GOTO "top"

! Exit
:E:$EXIT
```

FIGURE 2
Example "Subroutine" File

```
! HELP.MNU - DECUS Australia Symposium 1985

:TOP:$CLEAR
$DISPLAY 1 2 "HELP for "<lfile>" Option "hlp
$GOTO hlp

! WP.MNU HELP
:1:
    HELP for editing an existing file....
    (could use system help etc.)

$GOTO <HELP>
:2:
    HELP for creating a new file. ..

$GOTO <HELP>
:3:
    HELP for directory...

$GOTO <HELP>
:4:
    HELP for printing...

$GOTO <HELP>
:E:
    HELP for the EXIT function...

$GOTO <HELP>

! HELP for some other file. e.g.
:A:
:B:

! Wait for user input
:HELP:$DISPLAY 1 23 "Press any key to return to menu"
$INPUT 33 23 any

! Error traps and return
:ERROR:
:UP:
:C:$GOTO <lfile>
```

- the main body of the menu is specified in image text mode to reduce display overheads.

- option selection is via $BRANCH single character input.

- options 1 and 2 share the same $DCL command with different switches.

- subroutine INFILE is used for prompting by options 1,2 and 4.

- options 1 and 4 require confirmation that the requested file actually exists.

- option 4 requires confirmation of allowable printer via a valid printer lookup table.

- pressing PF2 at any stage of input will branch to the label HELP (the default contents of the permanent symbol <HELP>). An option is then requested on which to give help. the option is verified via $TESTLABEL and control is passed to the file HELP.MNU.

- trap labels are specified for :ERROR: and :^C:.

The major features of the "subroutine" help file HELP.MNU are:

- image text is used to describe each option.

- always returns control to <LFILE>, the last-used file name.

DPL LANGUAGE DEFINITIONS

Directives

All directives must be preceded by a dollar sign "$". Only one directive allowed per line. A directive may only be preceded on a line by a label or a series of tabs or spaces.

Labels

All labels must be prefixed and suffixed by a colon":". Only one label allowed per line. A label must be the first non-tab/blank character on a line. A label may be on a line by itself. Label names are limited to six characters (not including colons) cannot have imbedded blanks and should be unique.

Comments

Comment lines must have an exclamation point "!" as the FIRST character of the line.

Image Text

Any line without "!" as the first character or "$" or ":" as the first non-blank/tab character will be printed directly to the screen. (This does not apply if the lines are written in a table data definition region.)

Permanent Symbols

Any symbol enclosed in angle brackets refers to a DPL permanent symbol name. Permanent symbols may be $SET by the user but not $DELETEd. Permanent symbols include:

<DATE>	Current date ddd-mmm-yyy
<TIME>	Current time hh:mm
<^C>	System abort interrupt label
<ERROR>	Default error trap label
<UP>	General interrupt lable
<DOWN>	" " "
<HELP>	" " "
<SOFABO>	" " "
<ALTENT>	" " "
<MFILE>	Master file specification
<CFILE>	Current file
<LFILE>	Last used file
<TFILE>	Tree file
<NFILE>	Next-to-use file
<TERM>	Terminal type identifier 1=VT52, 2=VT100, 3=HAZELTINE, 4=ADM
<USER>	Username
<LINE>	Terminal line number
<INDEX>	$LOOKUP returned pointer value
<PROJ>	Project number of user
<PROG>	Programmer number of user

User Defined Symbols

Any six character string which does not start with "$" or ":" or contain a "+" or """ can be used as a user symbol. Each symbol can "store" up to 80 characters of data.

Symbol Concatenation Character

A "+" may be used to delimit any two user symbols in a string expression.

Relational Operators

The following relational test operators are permitted:

Test	Operator	
Equal to	EQ	or ==
Not Equal to	NE	/=
Greater than	GT	>>
Less than	LT	<<
Greater than or equal to	GE	>=
Less than or equal to	LE	<=

Syntax Item Definitions

NOTE: Tabs or spaces are the only valid item separa-tors.

Primitives:

symbol	- either a user defined (or to be defined) symbol name up to six characters in length e.g. FILE . - or a permanent symbol name e.g. <DATE>
tablename	- a user defined (or to be defined) symbol name to be associated with a lookup data table e.g. DEVTBL
label	- a user defined symbol name associated with a command in the file. e.g. :LOOP:
txt_string	- any characters contained by double quotes e.g. "this is a text string"
x y	- integer screen co-ordinates, top LH corner is origin = 1,1 e.g. 10 20
str_len	- integer max length of an $ACCEPT string.

Derivatives:

str_exp	- a string expression consisting of combinations of symbols and txt_strings.
default	- an initial str_exp value for an $ACCEPT symbol.
relop	- a str_exp containing a relational operator.
location	- a str_exp containing a label definition.
else_label	- a str_exp containing a label definition which will be used if the logical test is FALSE.
table	- a str_exp containing a table definition.
seconds	- a str_exp representing an integer number of seconds.

Directive Syntax

$ACCEPT x y str_len default symbol {flag}

Display the default string at column x, row y and allow editing of the default string up to a maximum of str_len characters. Will optionally accept an extra flag argument "silent" to prevent user input being echoed. Return the edited string as a symbol when either of the following terminators is entered.

<carriage-return>	- normal return, continue execution on next line of command file.
<system-abort>	- branch to interrupt label stored in the permanent symbol <^C> else abort if no label specified.
<up-arrow>	- branch to interrupt label <UP> if present, else ignore and continue $ACCEPT command.
<down-arrow>	- branch to interrupt label <DOWN> etc
<help>	- branch to interrupt label <HELP> etc
<soft-abort>	- branch to interrupt label <SOFABO> etc
<alternate-enter>	- branch to interrupt label <ALTENT> etc

```
e.g. $ACCEPT 10 10 14 "DY1:FRED.DAT" INPUT
     $ACCEPT 10 10 14  DEFAULT INPUT
     $ACCEPT 10 10 14  DEFAULT DEFAULT
```

$BRANCH_ON_INPUT x y symbol

Position the screen cursor at (x y) and wait for input of a single character. Store the single character in symbol and then execute a $GOTO symbol. Primary use

is for menu selection. If the symbol contents do not correspond to a label in the file a "beep" is issued and the $BRANCH is re-executed. All normal $ACCEPT terminators still operate.
e.g. $BRANCH 10 10 OPTION

$CLEAR {x y}

Clear from position x y to end of screen. Default x y is 1 2. (Top line of screen is reserved for DPL licence details).
e.g. $CLEAR
     $CLEAR 1 15

$DCL str_exp

Pass the nominated str_exp to the system monitor and exit from DPL. Before exiting DPL an implicit $SAVE will be executed and the chain-back command "RUN/S DPL:DPL /D" is inserted into the chain buffer to enable a return to the current file and line number with all symbols intact. To return to a different file use $SET <NFILE> "newfile" prior to $DCL. Any number of $DCL commands may be specified.
e.g. $DCL "RUN "PROGRA".BAS"
     $DCL "@"DEVICE"EDIT "INPUT" "ARG1" "ARG2 "ARG3"

$DELETE symbol

Remove the nominated symbol from the user symbol table and make its data space available for other symbols. Permanent symbols cannot be deleted.
e.g. $DELETE INPUT

$DISPLAY x y str_exp

Display the contents of the str_exp starting at column x, row y.
e.g. $DISPLAY 10 10 PROMPT<DATE>
     $DISPLAY 10 10 "SELECT"
     $DISPLAY 10 10 "SELECT "CHOICE+FILE" FOR"ACTION

$DEF_TABLE table

Defines the start of a data table definition. All following lines will be treated as literal data if the first non-blank/tab character is not a "$". The table data should be terminated by an $END_TABLE directive.
e.g. $DEF_TABLE DEVICE

$END_TABLE

Defines the end of any previously defined table.
e.g. $END_TABLE

$EXIT

Terminates processing and returns to DCL. Also, if used to terminate a string of $DCL commands it prevents the inclusion of the chain-back command "RUN /S DPL:DPL /D" into the DCL command string. It also prevents creation of DPL???.DAT save file during $DCL which means current DPL line number will not be saved.
e.g. $EXIT

$GOSUB label

Transfers execution to a subroutine of DPL commands within the current file. There are up to ten levels of nesting allowed and execution begins at the label specified and after a $RETURN will resume at the line following the $GOSUB which called it.
e.g. $GOSUB INPUT

$GOTO location

Transfer execution to the nominated location. if location does not substitute to a label in the file, assume it's a file specification.
e.g $GOTO LABEL
    $GOTO "LABEL1"
    $GOTO "LABEL"NUM
    $GOTO "DY0:MENU1.MNU"

$IF str_exp relop str_exp else_label

Perform the nominated relational test between the two str_exp and if the result is TRUE continue execution on the next command line. If the result is FALSE perform a $GOTO else_label.
e.g $IF ANS EQ "YES" NO
    $IF FILE NE OUTPUT".DAT" OUTPUT

$INPUT x y symbol {flag}

Perform same function as $BRANCH but simply store the single character input in symbol. Will optionally accept an extra flag argument "silent" to prevent user input being echoed.
e.g. $INPUT 10 10 OPTION

$LOOKUP str_exp table else_label

Perform a lookup of str_exp in the nominated data table and if successful continue execution on the next command line. The permanent symbol <INDEX> contains the table entry position where the lookup was successful. If the lookup fails, execution is transferred to else_label
e.g. $LOOKUP DEV1 DEVICE NODEV
     $LOOKUP "PD0:"DEVS "NODEV"
     $LOOKUP "PD1:"+INPUT".DAT" FILES NOGOOD

$RETURN

Marks the end of a subroutine section of DPL commands. Execution control is returned to the command line immediately following the $GOSUB command which invoked the subroutine. If no previous $GOSUB the command is ignored.
e.g. $RETURN

$SAVE

Indicates that all user symbol data will be retained after execution is passed to another file. If not specified all user symbols are cleared. Only applies to the current file.
e.g. $SAVE

$SET symbol str_exp

Store the contents of the string expression as the nominated symbol.
e.g. $SET DEFAULT "PD0:INPUT.TXT"
     $SET <ERROR> "ERRLAB"
     $SET <HELP> "HELP"
     $SET OFILE NAME+CODE".DAT"

$TESTFILE str_exp else_label

Determine if str_exp exists as a file in the system and if it does, continue execution on the next line. If it does not exist, $GOTO else_label.
e.g. $TESTFILE "PD0:MYFILE.TXT" NOWAY
     $TESTFILE OUTPUT".TXT" NOPE

$TESTLABEL label else_label

Checks to see if a label exists in the current file. branches to else_label if it doesn't.
e.g. $TESTLABEL INPUT NOLABL

$WAIT seconds

Suspend execution of the file for the integer number of seconds specified in the string expression seconds.
e.g. $WAIT "10"
     $WAIT PAUSE

CONCLUSION

DPL is an easy-to-use yet powerful command language in the style of IND and DCL designed for commercial applications under the TSX-Plus. RT-11 and even VMS operating systems. The language is under continued development with planned extensions to include a PARSE directive, extended GOTO capabilities, enhanced video controls. numeric symbols and a general file I/O interface.

REFERENCES

1. B. A. Harper,
"SPECS: A Menu Control System For RT-11",
Proc. Digital Equipment Computer Users Society,
Melbourne, Australia, July 1982.