Professional™
300 series

PRO/Office Workstation Programmer's Manual

Order No. AA-BM02A-TK

Developer's Tool Kit

digital
software

PRO/Office Workstation Programmer's Manual

Order No. AA-BM02A-TK

April 1984

This document describes how to customize the PRO/Office Workstation.

REQUIRED SOFTWARE:	PRO/Office Workstation V1.0 Professional Host Tool Kit V1.7 or later or PRO/Tool Kit V1.0 or later PRO/Communications V1.8 or later
TARGET OPERATING SYSTEM:	P/OS V1.7 or later
SUGGESTED CONFIGURATION:	Host VAX running VMS V3.4 or later ALL-IN-1 V1.3 or later Professional 350 connected to host via Hardware, Modem, Gandalf, or Micom Switch

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754

First Printing, April 1984

CONTENTS

TABLES

PREFACE

MANUAL OBJECTIVES

This manual describes how you can customize the PRO/Office Workstation by using software supplied with both the Tool Kit and PRO/Office Workstation.

INTENDED AUDIENCE

You should have some experience with the Professional Developer's Tool Kit, especially with the utilities FMS, and DCL. Also, you should have some experience programming on an RSX-11 or RSX-11/M-PLUS system.

STRUCTURE OF THIS DOCUMENT

The manual has eight chapters:

- **Chapter 1,** Programmer's Overview of PRO/Office Workstation, introduces the product. It describes the relationship between PRO/Office Workstation and the P/OS operating system. Also, the chapter provides details on PRO/Office Workstation as an application installed on the Professional 350.

- **Chapter 2,** Using the Flow Control Facility, describes the major component of PRO/Office Workstation. This chapter illustrates the ways that you can invoke Flow functions.

- **Chapter 3,** Flow Control Functions, lists and describes all the functions that Flow provides. We present the functions in alphabetical order, providing format, description, and examples for each.

- **Chapter 4,** Using the Form Interface Facility, describes the facility that handles all form processing for PRO/Office Workstation. This chapter describes how you display forms, as well as how you customize them.

- **Chapter 5,** Using the Document Services Facility, describes the facility that handles documents. We show you the organization of the database that Document Services uses, and we describe the special functions you can call to perform Document Services tasks.

- **Chapter 6**, Using the Symbol Services Facility, describes how PRO/Office Workstation interprets symbols and performs symbol substitution. Also, the chapter describes qualifiers you can use when defining or deleting symbols.

- **Chapter 7**, Using the Mail Services Facility, describes the functions you can use to send messages to a host VAX running ALL-IN-1.

- **Chapter 8**, Using the Network Services Facility, describes XCOM and XLIB, two tasks that manage communications between the Professional 350 and a host VAX.

ASSOCIATED DOCUMENTS

- PRO/Tool Kit Command Language and Utilities Manual
- Tool Kit User's Guide
- P/OS System Reference Manual

CONVENTIONS USED IN THIS DOCUMENT

This document uses the following conventions.

Convention	**Meaning**
[optional]	Square brackets indicate optional fields in a command line. Do not include the brackets in the command line. If the field appears in lowercase type, you must substitute a legal parameter if you include the field. Do not confuse the square brackets in a command line format with the square brackets that you use to specify a file specification.
UPPERCASE	Any command line field in uppercase type indicates that you should type the word or letter exactly as shown.
lowercase	You must substitute a value for any command line field in lowercase type. Usually the lowercase word identifies the type of substitution required.

Convention	Meaning
...	A horizontal ellipsis indicates that you can repeat the preceding item one or more times. For example: parameter [,parameter...]
. . .	A vertical ellipsis in a figure or example means that not all of the statements are shown.
red ink	Indicates user input in examples.

Additionally, note that numeric values are decimal unless specified otherwise.

CHAPTER 1

PROGRAMMER'S OVERVIEW OF PRO/OFFICE WORKSTATION

The PRO/Office Workstation provides standard office automation functions on the Professional 350. Like DIGITAL's 32-bit ALL-IN-1 running on VAX, PRO/Office Workstation offers a consistent, yet easily customized, user interface.

The main feature of PRO/Office Workstation is the local processing and storage capability provided by the Professional 350. Some advantages of this feature are:

- Greater host CPU efficiency gained by local processing (especially when running high-overhead tasks like text editors).
- Consistent local system response regardless of the number of users logged in to the host.
- Reduction of storage requirements on the host due to local storage.
- Greater system security for documents that are locally stored.
- Ability to run local applications.

When combined with ALL-IN-1 running on VAX (as a host computer), PRO/Office Workstation provides these advantages, as well as the traditional office automation features.

This chapter introduces PRO/Office Workstation by describing its relationship with P/OS, as well as its components and structure.

1.1 PRO/OFFICE WORKSTATION AND PROFESSIONAL 350 ARCHITECTURE

PRO/Office Workstation is a program that runs under the P/OS operating system. It is an application that users install like any other application.

However, unlike a typical application, PRO/Office Workstation is capable of interconnecting and integrating other applications and services that draw upon the resources of the Professional 350. Figure 1-1 shows the Professional's resources in terms of layers. Each layer builds upon the previous one. The uppermost layer is PRO/Office Workstation.

In the figure, intersecting lines connect the PRO/Office Workstation layer with underlying P/OS layers. These lines illustrate the flexibility that allows users to access the Professional's resources in many different ways. This flexibility arises from a concept that is central to the implementation of PRO/Office Workstation: the separation of form and function.

1.1.1 Separation of Form and Function

As an example of the separation of form and function, consider the various ways that a user might run a task. In each case, the function is the same, but the form is different.

- Invoke the RUN function from PRO/Office Workstation's command language interpreter.
- Press a function key that has been defined to invoke the RUN function.
- Execute a command procedure that invokes the RUN function.
- Enter a user-defined (dynamic) command that invokes the RUN function.
- Select it from a PRO/Office Workstation menu.
- Select it from your own, custom-designed menu.

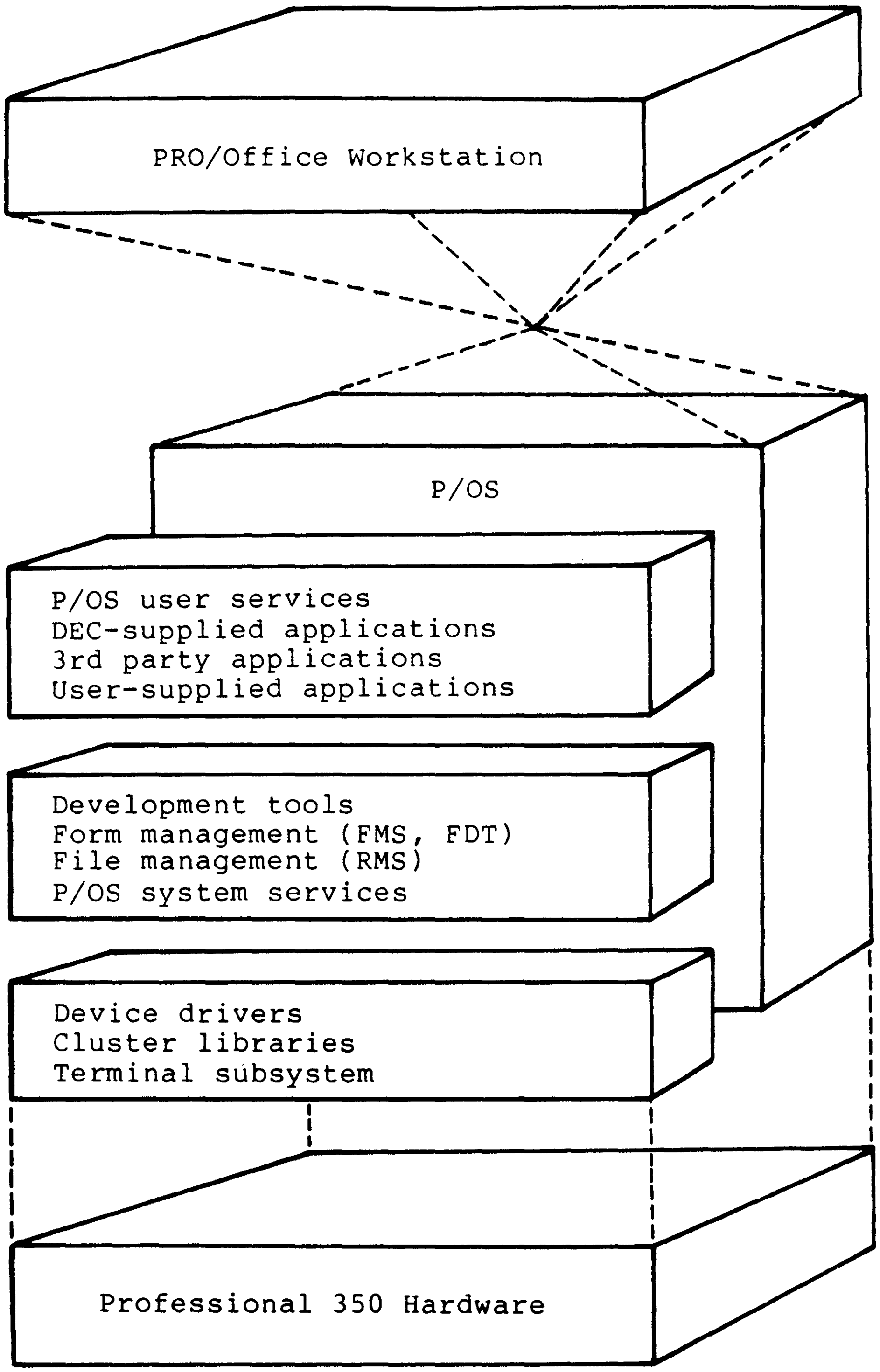


Figure 1-1: Layers of Professional 350 Resources

The last two examples in the previous list interpret the term 'form' in a literal sense. Literally, a form is a display of information for the user, like a menu or a help frame.

PRO/Office Workstation uses the Professional's form utilities -- FDT and FMS -- to create, store, modify, and display forms. You can use these utilities to create forms that look the way you want, without being tied to the functions they perform.

1.1.2 Organization of PRO/Office Workstation

The end-user interface of PRO/Office Workstation consists of subsystems. A subsystem is the sum of PRO/Office Workstation facilities that perform a task, such as processing a document or or sending a mail message. The first four options on the Main Menu constitute the PRO/Office Workstation subsystems:

1. Document Processing
2. Electronic Mail
3. Desk Management
4. Business Applications

While a subsystem is defined by a Main Menu option, a facility is defined by its single functional capability. A facility is a collection of PRO/Office Workstation software modules that perform a single function. One subystem might invoke several facilities. For example, the Electronic Mail Subsystem invokes both the Form Interface Facility and the Mail Services Facility. It is also true that several subsystems can invoke one facility. For example, all the subsystems invoke the Form Interface Facility.

Table 1-1 lists and briefly describes the PRO/Office Workstation facilities.

Table 1-1: PRO/Office Workstation Facilities

Facility	Description
Flow Control	The "operating system" of PRO/Office Workstation. It provides access to all the other facilities. Flow translates a subsystem request for a facility into a function that the facility understands. We describe these functions in the chapter on Flow Control.
Form Interface	Manages the display and processing of forms, including menus, from which novice users can access functions performed by Flow. Controls the flow of data into and out of fields -- areas of forms through which interchange of data takes place. Interprets special keyboard keys that the user presses while in a field. The Form Interface Facility uses the Forms Management System (FMS) and the Frame Development Tool (FDT).
Document Services	A database management system that provides functions to create, store, file, and modify documents.
Symbol Services	Defines and translates all symbols in PRO/Office Workstation.
Mail Services	Uses Document Services to create and store messages, and uses Network Services to send and recieve messages from remote systems.
Network Services	Manages all communications between the Professional 350 and other computers. Controls all operations involving the Professional's Communication Port, XK0:.

As described in Table 1-1, the Flow Control Facility is the "operating system" of PRO/Office Workstation. It consists of a set of software modules that perform functions and process requests for the other PRO/Office Workstation facilities.

Figure 1-2 illustrates this organization. The figure shows a detail of PRO/Office Workstation as one of the layers of Professional 350 resources.

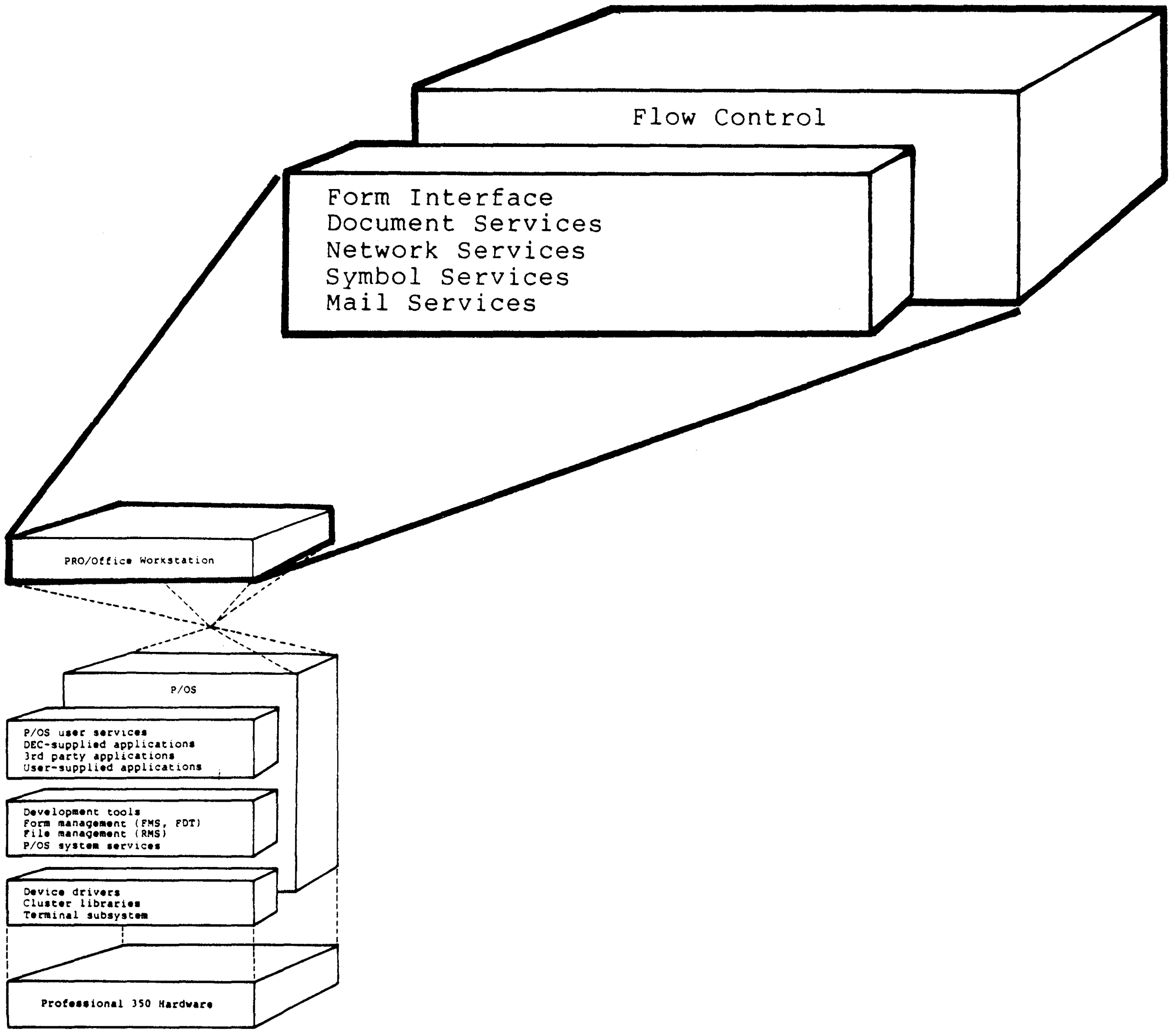


Figure 1-2: The PRO/Office Workstation Layer

The remainder of this chapter describes PRO/Office Workstation as an application installed on the Professional 350.

1.2 PRO/OFFICE WORKSTATION AS AN APPLICATION

As mentioned earlier in this chapter, PRO/Office Workstation is an application that users install on the Professional 350. The following sections provide details about the application. Note that the general information in the following sections pertains to all applications; only details such as application filenames and the contents of the installation file are unique to PRO/Office Workstation.

1.2.1 PRO/Office Workstation Installation Files

The application diskettes contain directories and files that P/OS uses to install and execute PRO/Office Workstation. The main application directory, as well as the installation file it contains, uses the PRO/Office Workstation application name, OA:

```
[OA]OA.INS
```

If you use disk drive 1, you can search for this file on the application diskettes by entering the Tool Kit/DCL command:

```
$ DIR DZ1:[*]*.INS
```

When installation begins, P/OS searches the diskettes for this directory and .INS file. Then P/OS:

- displays the application name,
- allows the user to change this name and specify a menu on which to place it, and
- places the name on the requested menu.

Next, P/OS creates a directory and an empty file, to which it copies the application directory and installation file. P/OS names these:

```
[ZZAPnnnnn]ZZAPnnnnn.INS
```

Where:

nnnnn	is a zero-filled, five-digit decimal integer representing the number of currently installed applications plus one.

For example, if eight applications are currently installed, P/OS uses 00009 for n and creates:

[ZZAP00009]ZZAP00009.INS

P/OS maintains a table of installed applications in a file called [ZZSYS]INSAPPL.SYS. During every installation, P/OS updates the file. (Note that the names of all system directories on P/OS begin with the letters ZZ.)

As the last step, P/OS reads the installation file. This file directs the transfer of the other application files from application diskettes to hard disk. P/OS performs the transfer.

The installation file uses special keywords that direct P/OS during the installation:

NAME	Specifies the default application name as it first appears to the user during installation. The name can be as long as 40 characters, and can contain any printable characters and spaces.
FILE	Specifies the name of a file for P/OS to copy from the diskette onto the hard disk. The /DELETE qualifier and the /KEEP qualifier (not shown) determine whether or not P/OS deletes the specified file if the user ever removes the application.
MOUNT	Specifies the volume name of an additional diskette for applications contained on multiple diskettes. The MOUNT line determines whether or not the user must insert the diskette during installation (it might already be in the other disk drive).
INSTALL	Indicates a task image (/TASK) or resident library (/LIBRARY) that P/OS installs whenever the user invokes this application. On P/OS, a task can run only if it is installed. Also, a resident library must be installed in order for a task to refer to it.
ASSIGN HELP	Assigns a default help definition file and frame ID to the application. The application uses this information to display FDT help messages when the user presses the HELP key.

RUN	Specifies the first task that executes when the user invokes the application.

The installation diskettes contain files in the following directories:

- [ZZAP...]

  The main application directory, whose name P/OS converts from [OA]. This directory contains only the installation file (.INS).

- [ZZFLOW]

  Contains all tasks (except those belonging to Network Services), form libraries, and command procedures.

- [ZZDOC0]

  Contains Mail Services and Document Services files, including the default document database, mail queues (folders), message templates used by Mail and Document Services, and initialization files for the available text editors.

- [ZZOASYM]

  Contains the system, process, and user symbol tables maintained by the Symbol Services Facility.

- [ZZXNET]

  Contains tasks and other files that the Network Services Facility uses.

Table 1-2 describes the task images contained in the installation diskettes. Table 1-3 describes other installation files.

Table 1-2: PRO/Office Workstation Installation File Tasks

Filename	Installed Name	Description
CDS.TSK	SWB$DS	Callable Document Services, a component of the Document Services Facility.
DSI.TSK	F$DSI	Document Services Interface, a component of the Document Services Facility.
EDT.TSK	EDT	The EDT editor, also used for WPS.
FBOOT.TSK	FBOOT	Task that enables or disables automatic bootstrap into PRO/Office Workstation.
FCSRES.TSK	FCRES	Resident library used by PIP and EDT.
FI.TSK	SWB$FI	The Form Interface Facility.
FLOW.TSK	FLOW	The Flow Control Facility.
FMAIL.TSK	F$MAIL	Flow Mail, a component of the Mail Services Facility.
FSYM.TSK	FSYM	Code executed for the Flow function SHOW SYMBOL.
PIP.TSK	PIP	Peripheral Interchange Program that performs file control functions for Flow.
SB.TSK	SWB$SB	Part of the Symbol Services Facility.
SETVT.TSK	SETVT	Task that sets the terminal to VT100 mode. It is useful when you execute FED from Flow command level.
TMAIL.TSK	SWB$BM	Transport-Level Mail, a component of the Mail Services Facility.
TYPE.TSK	TYPE	Task that Flow invokes for SHOW FILE and TYPE functions.

Filename	Installed Name	Description
XCOM.TSK	XCOM	Port controller task in the Network Services Facility.
XLIB.TSK	XLIB	Path maintenance task in the Network Services Facility.

Table 1-3: Other PRO/Office Workstation Installation Files

Filename	Description
DEFDOC.COM	Defines initial document databases and related symbols.
DEFSYM.COM	Defines default dynamic commands, function kyes, symbols, and tags.
DEFUSER.COM	Defines user-oriented values, such as profile information and preferred editor.
EDTINI.EDT	Initialization file used by EDT. See the EDT command in Chapter 3.
EDTSYS.EDT	Initialization file used by EDT. See the EDT command in Chapter 3.
GETRMS.COM	Command procedure that reads a record from a specified RMS file and writes it to an existing document file. Used for file cabinet maintenance.
INSPROCOM.COM	Copies the special PRO/Communications tasks into DW1:[ZZCOMM].
INSTALL.COM	First command procedure that Flow executes after you have installed PRO/Office Workstation.
NOTES.MEM	Release notes for the current version of the product.
OAFDT.FLB	Default FDT form library.
OAFMS.FLB	Default FMS form library for all PRO/Office Workstation forms.
OAHELP.FLB	Default help frame library.

Filename	Description
PUTRMS.COM	Command procedure that writes an existing document file to a specified RMS file. Used for file cabinet maintenance.
SETBOOT.COM	Command procedure that calls FBOOT.TSK to enable automatic bootstrap of PRO/Office Workstation.
SETNOBOOT.COM	Command procedure that calls FBOOT.TSK to disable automatic bootstrap of PRO/Office Workstation.
SYMBOL.DAT	Part of the Symbol Services Facility.
WPSINI.EDT	Initialization file used to make EDT look like WPS. See the EDT command in Chapter 3.

1.2.2 Installation Startup Procedure

The first procedure that Flow executes after you have installed the kit is [ZZFLOW]INSTALL.COM. This file performs initialization for the Workstation. You should never modify this file.

INSTALL.COM executes several command procedures, which you can re-execute at any time to re-initialize the workstation. Also, you can modify any of these procedures to customize the installation. The procedures are:

- **DEFDOC.COM**

  This procedure sets up the default document database for the Document Services Facility. It also defines some symbols that Document Services uses.

- **DEFSYM.COM**

  This procedure defines all of the default Workstation symbols, including dynamic commands, function keys, tags, and symbols. You might want to rerun DEFSYM.COM if a user inadvertently redefines any reserved symbols.

- **DEFUSER.COM**

  This procedure defines user-specific values, such as information that is part of the user profile.

- **INSPROCOM.COM**

  This procedure copies certain PRO/Communications tasks that the workstation uses into the directory DW1:[ZZCOMM].

Once installed, the Workstation does not require a startup command file; however, you can easily define your own startup command file by by defining the symbol FLOW$_STARTUP_FUNCTION.

For example:

```
> DEF/SYM FLOW$_STARTUP_FUNCTION "@[COMMANDS]LOGIN.COM"
```

Flow would execute this file every time the user invokes the PRO/Office Workstation application. LOGIN.COM could define various symbols, tags, and logical names, and could perform other startup processing.

1.2.3 Storage Requirements

Without any customization, PRO/Office Workstation requires approximately 2000 contiguous blocks. You can reduce the storage requirement by purging the system after installing the application:

```
> PURGE [*]*.*
```

NOTE

Purge the entire system only if you are certain that the user needs only the latest versions of all files on the system. A system purge can delete important files.

The purge essentially removes files duplicated by PRO/Office Workstation. For example, the system might have the PRO/Tool Kit installed, in which case the purge deletes earlier versions of those Tool Kit tasks that are shared with PRO/Office Workstation.

1.2.4 Finding the Electronic Release Notes

PRO/Office Workstation provides release notes for every version; these notes ensure that you are informed of any changes to the product that could not be included in this manual.

You should always read the release notes. Print the following file directly from the installation diskettes:

[ZZDOC0]NOTES.MEM

The installation procedure copies this file to a document in the default Document Services database. You can look at the document by finding

Folder: RELEASENOTES
Title: RELEASE NOTES version

Where:

version	is the product's current version number.

CHAPTER 2

USING THE FLOW CONTROL FACILITY

The Flow Control Facility ("Flow") directs all activity in the PRO/Office Workstation environment. Flow is the workstation's "operating system." It consists of a set of software modules that perform functions and process requests for the other PRO/Office Workstation facilities.

You can directly access Flow through either of its user interfaces:

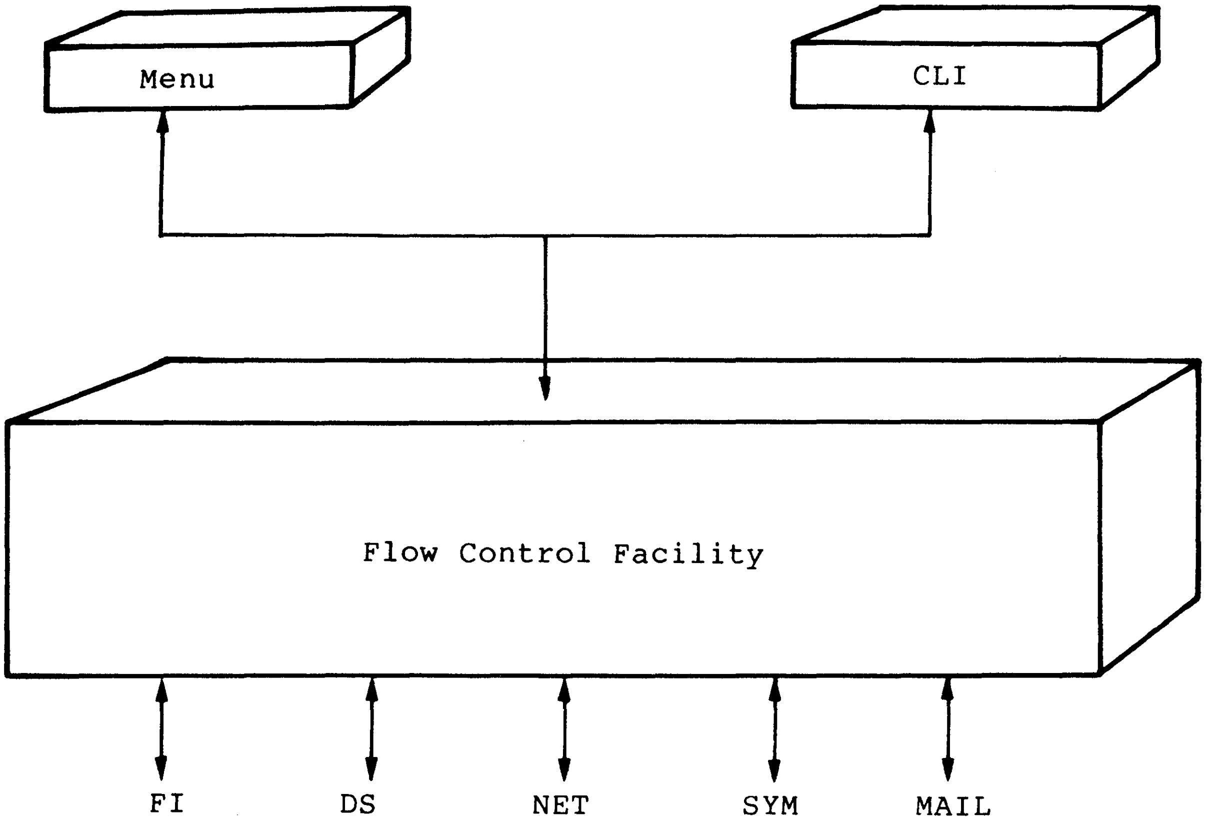
- **Flow Menu System**

  The Menu System allows a novice user to request Flow functions, such as running applications, by selecting items from a menu. By default, all menus are FMS forms. By making a menu selection, a user effectively sends FMS named data to Flow for translation into Flow functions.

- **Flow CLI**

  The CLI is a command language interpreter that is similar in purpose and syntax to the DIGITAL Command Language (DCL). Each CLI command represents a Flow function. The CLI is designed as an interface for expert users.

Figure 2-1 illustrates the operation of Flow. Users access Flow's functions through either the menu or CLI interface. Flow processes the request by performing the function itself or by passing the request on to another facility. The results of the request travel back to the user interface, as shown by the two-way arrows.



FI = Form Interface Facility
DS = Document Services Facility
NET= Network Services Facility
SYMS = Symbol Services Facility
MAIL = Mail Services Facility

Figure 2-1: The Flow Control Facility

Flow's different user interfaces allow a user to choose the most appropriate means of invoking Flow functions. A first-time user might use the menu structure initially. After gaining experience on the system, however, the user can invoke one-time Flow functions from the PRO/Office Workstation Main Menu, or can directly enter CLI command mode by pressing the INSERT HERE key from any menu.

In general, however, the application programmer uses the Flow Control CLI to invoke Flow functions.

2.1 USING THE FLOW CONTROL CLI

The Flow Control CLI looks very much like a subset of DCL. The format of CLI commands is often the same as DCL commands. This section summarizes CLI format.

NOTE

> CLI commands are Flow functions. We use the terms 'CLI command' and 'Flow function' interchangeably.

A CLI command consists of a command name that describes the action Flow is to perform. Additionally, most commands include one or more qualifiers and parameters to further define Flow's action. The qualifiers can themselves take arguments.

The general format of a CLI command is:

```
[$] [label:] command[/qual[=arg]...]  [param[/qual[=arg]]...]...
```

Where:

$	is the dollar sign symbol, which you should place at the beginning of each command line that is in a command procedure. If you specify the $ symbol at the beginning of a command line entered interactively, CLI ignores the symbol.
label:	is the name of a point to which you can transfer control within a command procedure. You use the GOTO statement to transfer control.
command	is a Flow function.
qual	is a valid qualifier for the particular command or parameter.
arg	is a valid argument for a particular qualifier.

As the format shows, a slash (/) always precedes a qualifier, a space always precedes a parameter, and an equal sign (=) always precedes an argument. You can use a tab or multiple spaces wherever you can use a single space in a command line.

Some commands require that you include a qualifier, parameter, or argument on the command line. If you fail to supply a required command element, Flow prompts you for that required element. (In some cases an omission causes an error rather than a prompt.)

You can enter CLI commands that are minimally-unique; that is, you need only type the abbreviated part of the command that distinquishes it from any other CLI command. For example, you can abbreviate the DIRECTORY command as DIR. Also, you can abbreviate DELETE/CONFIRM as DEL/CONF.

2.1.1 Symbol Substitution in Command Line

Flow attempts to parse all symbol references. A symbol reference can be any of the following:

- A character string beginning and ending with an apostrophe, such as '$RESULT'.
- A character string that you use in a context within which Flow expects a symbol value to appear, such as the IF, INQUIRE, WRITE, and LET functions in Flow. In these cases, you do not use apostophes to force the translation.

A symbol reference can appear anywhere within a Flow command line, whether the command line appears interactively on the terminal screen or within a command procedure. Flow passes a symbol reference to the Symbol Services Facility for translation into the equivalence value, if any.

Whenever you want to parse and translate a symbol, whether or not it is within a quoted string, you must place one apostrophe on each side of the symbol (except when Flow expects a symbol to appear):

```
> DEFINE/SYMBOL SYMISCOMMAND  "DIR"
> 'SYMISCOMMAND'                          ! Translates to DIR

> DEF/SYMBOL ANOUN "NAME?"
> INQUIRE NAME  "ENTER 'ANOUN?'"          ! Prints ENTER NAME?

> DEF/SYMBOL ANOUN "NAME?"
> WRITE SYS$OUTPUT ANOUN                 ! Prints NAME?
> WRITE SYS$OUTPUT "ANOUN"               ! Prints "ANOUN"
```

If you want to include apostrophes or quotation marks in a quoted string, then double them.

```
> INQUIRE NAME "ENTER ''ANOUN''"          ! Prints ENTER 'ANOUN'
> INQUIRE NAME  "ENTER ""ANOUN"""         ! Prints ENTER "ANOUN"
```

An example of the LET functions follows.

```
> LET VALUE=10
> LET TOTAL=VALUE+2
```

2.1.2 Commands Grouped by Purpose

Table 2-1 shows all the CLI commands grouped by purpose. Chapter 3 lists the commands alphabetically and describes each in detail.

Table 2-1: CLI Commands Grouped by Purpose

Purpose	Name	Description
Execution Control	CALL	Invoke one of PRO/Office Workstation's callable facilities.
	CONTINUE	Noop
	END	Exit the FLOW task.
	EXIT	Exit from the current Flow function.
	EXTERNAL	Invoke a PRO/Office Workstation facility that is callable only from Flow.
	GOTO	Transfer execution to a labeled statement in a command procedure.
	IF ... THEN	Conditionally execute a function.
	ON ... THEN	Establish a trap for a specified condition.
	UNWIND	Exit stacked FLOW functions until reaching the top (first) function.
Procedure Invocation	COMMAND or At sign (@)	Enter command mode, accepting commands from the terminal or from a command procedure.

Purpose	Name	Description
Symbol Handling	DEFINE/SYM	Create a symbol table entry and assign an equivalence name to it.
	DELETE/SYM	Delete a symbol definition from the symbol table.
	INQUIRE	Interactively assign a value to a symbol.
	LET	Assign the value of an expression to a symbol.
	SHOW/SYM	Display a symbol definition in a symbol table.
Keyboard Handling	DEFINE/KEY	Assign a Flow function to a specified key.
	DELETE/KEY	Remove a key definition established with DEFINE/KEY.
	SET [NO]KEYPAD	Enable or disable escape sequence recognition from the keyboard.
	SHOW KEY	Show one or all function key definitions.
Dynamic Command Handling	DEFINE/COM	Define a dynamic Flow command.
	DELETE/COM	Delete a dynamic Flow command.
	SHOW COM	Show one or all dynamic Flow commands.
Tag Handling	DEFINE/TAG	Define a Form Interface tag.
	DELETE/TAG	Delete a Form Interface tag.
	SHOW TAG	Show one or all Form Interface tags.
Task Handling	ABORT or STOP	Connect to and then abort a task.

Purpose	Name	Description
	ACTIVATE	Instruct P/OS to request a task.
	BLOCK	Stop the issuing task.
	BLOCK/SUSPEND	Suspend the issuing task.
	EMIT	Emit status.
	FIX	Load and lock an installed task or region in memory.
	INSTALL	Install a task or library.
	REMOVE	Remove (uninstall) a task or library.
	RUN/TASK	Run a task, installing it beforehand and removing it afterwards if necessary.
	UNBLOCK	Unstop a task that stopped itself with BLOCK.
	UNBLOCK/RESUME	Resume a task that suspended itself with BLOCK/SUSPEND.
Application Handling	DEFINE/APPL	Define a symbol as a P/OS installed application.
	DELETE/APPL	Delete a symbol defined as a P/OS installed application.
	RUN/APPL	Run a P/OS installed application that you have defined.
	SHOW APPL	Show one or all currently defined applications.
Form Handling	FIELD	Input or output one field of a form.
	FORM	Display a specified form.
	MENU	Display a specified menu.
Document Handling	DOC	Invoke a Document Services function.

Purpose	Name	Description
Mail Handling	MAIL	Invoke a Mail Services function.
Network Services	NETWORK	Invoke a Network Services function.
Disk/File Handling	COPY	Create a copy of one or more files.
	DELETE/FILE	Delete local files.
	DIRECTORY or SHOW DIR	Display information for an individual file or group of files.
	PURGE	Delete all but the latest versions of files, and releases the storage that they occupied.
	RENAME	Change the name, type, or version number of an existing file.
	SHOW DEFAULT	Display the current device and directory name.
	TYPE or SHOW FILE	Display a file.
Mis-cellaneous	ASSIGN	Associate a logical name with a physical device name, a complete file specification, or another logical name.
	CLEAR	Erase the terminal screen.
	CREATE/DIR	Create a directory.
	DEFINE/LOG	Define a P/OS logical name.
	DELETE/DIR	Delete a directory.
	DELETE/LOG	Delete a P/OS logical name.
	DISMOUNT	Declare a volume to be offline.

Purpose	Name	Description
	EDT	Run the EDT editor.
	HELP	Display a help screen.
	MOUNT	Declare a volume to be online.
	SET [NO]VERIFY	Control whether or not Flow displays command lines during command procedure execution.
	SHOW LOGICAL	Show the equivalence value of a logical name.
	WAIT	Suspend Flow for a number of seconds.
	WRITE	Print a string or symbol equivalence on the terminal screen.

You can enter CLI commands interactively by typing a command while in CLI mode, or you can store CLI commands in a command file and then execute the command file. The following sections describe both forms of entering CLI commands.

2.1.3 Using CLI Interactively

From the Main Menu you can invoke a single CLI command by entering a dollar sign followed by the command, and then pressing the RETURN key. To enter CLI command mode, simply press the INSERT HERE key.

Once you receive the CLI prompt, you can enter CLI commands at the keyboard. To execute a command, press either the RETURN or DO key. Since Flow waits for a carriage return before processing the command, you can use the DELETE key to edit the command line before pressing RETURN or DO.

The maximum number of characters in a command line is 132. However, you can enter the continuation character, a hyphen (-), at the end of a command line to continue onto the next line.

2.1.4 Using CLI Command Procedures

A CLI command procedure is a text file containing CLI commands. Flow reads the commands directly from the file, as if they were entered from CLI command mode.

You can initiate a command procedure by either using the COMMAND function or by using the at (@) sign before the filename. For example, from CLI you can enter either:

```
> COMMAND procedurename !  use COMMAND function
```

or

```
> @procedurename         !  use at sign
```

The procedure invocation can appear anywhere that a Flow function is allowed: for example, from the named data section of an FMS form, stored in a dynamic command using DEFINE/COMMAND, or stored in a keyboard function key using the DEFINE/KEY function.

- **In Named Data:**

Name	Data
INP	COMMAND procedurename

- **In a Dynamic Command:**

```
> DEFINE/COMMAND  GO  "COMMAND procedurename"
```

- **In a Function Key:**

```
> DEFINE/KEY 17 "COMMAND procedurename"
```

The default file type for command procedures is COM. Thus, the following two command procedure invocations are the same:

```
> @TEST.COM
> @TEST
```

Note that if you merely specify a file name, without preceding it with a directory, Flow searches the current directory only.

Each CLI command line within a command procedure except for continuation lines must begin with a dollar sign symbol ($):

```
$ command          ! line begins with $
command            ! no $ on line, therefore invalid
```

You can use a hyphen in a command procedure to continue a command onto subsequent lines. However, you must not precede a continuation line with the dollar sign. For example:

```
$ WRITE -
    SYS$OUTPUT -
    "THIS IS AN EXAMPLE"
$ EXIT
```

You can use the exclamation mark (!) to delimit comments within your command procedure. When a comment begins a line, you must still specify the dollar sign:

```
$ DIRECTORY    !Comment after CLI command
$ ! A full-line comment
```

The nesting level for command procedures depends on the amount of available stack space.

Normally, a command procedure executes synchronously; that is, Flow attaches the terminal. However, you can force Flow to process the command procedure asynchronously (in the background) by defining a dynamic command such as SPAWN:

```
> DEF/COM SPAWN "RUN/TASK/GO [ZZFLOW]FLOW.TSK -
                /NAME=SPAWN/CMD=""FLOW @"
```

Having defined the SPAWN command, you can execute a command procedure in the background by specifying:

```
> SPAWN procname
```

Where:

procname	is the name of your command procedure. It must not include the at (@) sign or the COMMAND function.

The dynamic command SPAWN operates by starting up a new Flow task, called SPAWN (/NAME=SPAWN). When translating your invocation of SPAWN, Flow automatically substitutes the procedure name after the /CMD=""FLOW @ portion of the command's equivalence string. The double quotes denote the beginning of the /CMD= string; the final single quote matches the single quote before the RUN command.

The command procedure you execute in this manner must not run a task that attaches the terminal, or you receive an error message.

Note that you do not have to specify the END statement at the end of your spawned command procedure; Flow automatically terminates itself.

2.2 THE FLOW STACK

Whenever you execute a Flow function, Flow pushes the function invocation onto the Flow stack. The <u>Flow stack</u> is an area of memory containing a list of Flow function invocations. Flow maintains the stack to preserve the order of functions that you have invoked.

Flow pops function invocations from the Flow stack when:

- You issue the EXIT, END, or UNWIND functions.
- The current function completes.

EXIT causes Flow to pop the current invocation from the stack, resulting in the previous invocation becoming the current one (except when the current invocation is already the first one). END causes Flow to pop all invocations from the stack, terminating Flow itself. UNWIND causes Flow to pop all but the first invocation from the stack. The first invocation becomes the current one.

Flow pops an invocation from the stack when the function completes. Note that some functions, such as MENU, do not complete unless the user explicitly uses EXIT, END, or UNWIND to pop them from the stack. This is useful, for example, if you ever have to trace a menu tree.

Figure 2-2 illustrates how the Flow stack operates. The figure shows successive changes to the stack as a result of function invocations:

1. The current stack contains four invocations in the order: MENU, MENU, MENU, and COM. Assume that the user invoked these functions from menus, and that the COM function places the user in the CLI.

2. During execution of the RUN function, the stack has a new invocation, RUN/TASK. This invocation becomes the current invocation.

3. Immediately upon completing the RUN/TASK request, Flow pops that invocation from the stack.

4. The EXIT function pops one MENU off the stack.

5. The UNWIND function pops subsequent invocations off the stack until Flow reaches the first invocation, MENU. Flow makes this the current invocation.

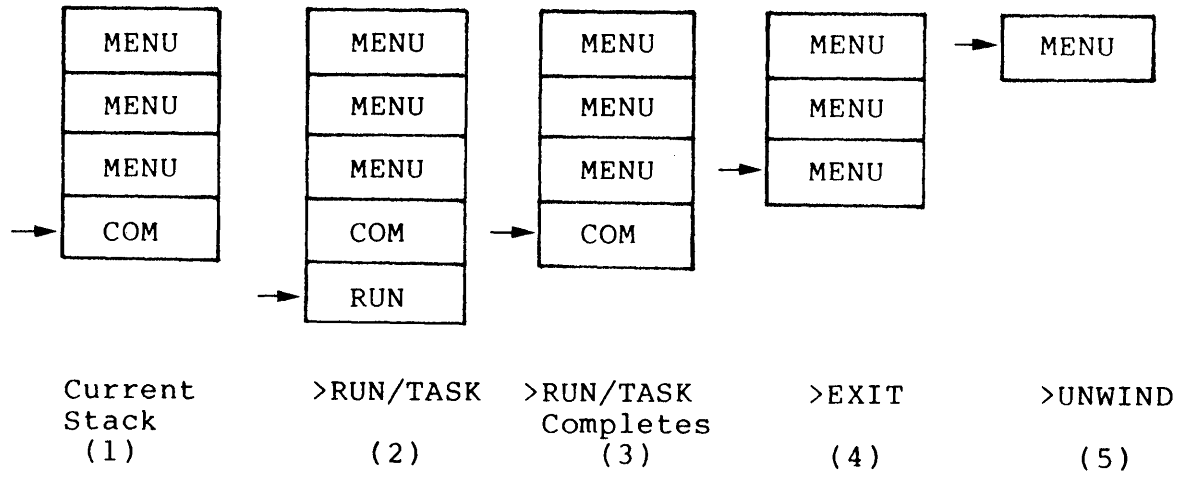


Figure 2-2: The Flow Stack

Note that in the figure the arrow (->) is the stack pointer. It indicates the current function invocation.

2.3 FLOW SYMBOLS

Table 2-2 describes symbols that Flow uses.

Table 2-2: Symbols Used by Flow

Symbol	Description
FLOW$_STARTUP_FUNCTION	Function executed to perform startup (normally is "CONTINUE").
FLOW$_COM_ACCOUNT	Filespec (dev:[dir]) of directory containing command procedures.
FLOW$_EXE_ACCOUNT	Original application directory (APPL$DIR in P/OS).

FLOW SYMBOLS

Symbol	Description
$RESULT	Contains the Flow function executed from FMS named data or FDT action string when you execute MENU/RETURN. Also can contain the exit status of a task that has just executed.
$STATUS	Contains the Flow status of the last executed Flow function.
$CHOICE	User's actual input after executing MENU/RETURN.
$NAME	Name of the task last installed.

CHAPTER 3

FLOW CONTROL FACILITY FUNCTIONS

This chapter describes all the Flow functions, presenting them in alphabetical order:

ABORT OR STOP
ACTIVATE
ASSIGN
BLOCK
BLOCK/SUSPEND
CALL
CLEAR
COMMAND
CONTINUE
COPY
CREATE/DIRECTORY
DEFINE/APPLICATION
DEFINE/COMMAND
DEFINE/KEY
DEFINE/SYMBOL
DEFINE/LOGICAL
DEFINE/TAG
DELETE/APPLICATION
DELETE/COMMAND
DELETE/DIRECTORY
DELETE/KEY
DELETE/SYMBOL
DELETE/LOGICAL
DELETE/TAG
DIRECTORY OR SHOW DIRECTORY
DISMOUNT
DOC
EDT
EMIT
END
EXIT
EXTERNAL
FIELD

FIX
FORM
GOTO
HELP
IF
INQUIRE
INSTALL
LET
MAIL
MENU
MOUNT
ON
PURGE
REMOTE
REMOVE
RENAME
REQUEST
RUN/APPLICATION
RUN/TASK
SET DEFAULT
SET KEYPAD
SET VERIFY
SHOW APPLICATION
SHOW COMMAND
SHOW DEFAULT
SHOW KEY
SHOW LOGICAL
SHOW SYMBOL
SHOW TAG
TYPE OR SHOW FILE
UNBLOCK
UNBLOCK/RESUME
UNWIND
WAIT
WRITE SYS_$OUTPUT

3.1 GLOBAL QUALIFIERS

Several qualifiers are global, that is, you can specify them on any Flow function. Two of these qualifiers, /CLEAR[=mode] and /[NO]PAUSE, affect the screen mode, which can be either:

- **Menu Mode**

  The user invokes Flow functions via menus displayed on the screen.

- **Command Mode**

  The user invokes Flow functions directly through the CLI.

The global qualifiers are:

/CLEAR[=mode]

Where:

mode	can be either MENU or CMD, indicating which screen mode the user <u>is about to enter</u>.

If you do not specify mode, then this qualifier refreshes the terminal screen before and after executing the function.

If you do specify mode, then this qualifier refreshes the terminal screen as appropriate for the mode the user is about to enter. For example, suppose you invoke the following command:

```
> DIR/CLEAR=CMD
```

The /CLEAR=CMD qualifier indicates that you are entering command mode. If you are currently in menu mode, Flow clears the screen before executing the DIR, and redraws the menu upon returning to menu mode. If you are already in command mode, Flow ignores the qualifier.

As another example, suppose you invoke the following command:

```
> MENU/CLEAR=MENU MAIN
```

If you are currently in command mode, Flow clears the screen before executing the MENU command. If you are already in menu mode, Flow ignores the qualifier.

/[NO]PAUSE

This qualifier causes Flow to pause and prompt the user to press any function key to continue. How the qualifier operates depends on the screen mode:

- If you are going from command mode to screen mode and PAUSE is set, then Flow will pause. If NOPAUSE is set, Flow will not pause.

- If you are going from menu mode to command mode, PAUSE and NOPAUSE have no effect.

Flow implicitly sets PAUSE whenever you are in command mode and you write to SYS$OUTPUT or SYS$COMMAND.

Flow implicitly sets NOPAUSE whenever you are in command mode and you read from SYS$INPUT or SYS$COMMAND.

/WARNING

Declare a WARNING condition, rather than an ERROR condition, if the Flow function is within a command procedure and it fails.

/QUIET

Do not print error message if the Flow function fails. Some Flow functions are not affected by this qualifier because they call P/OS utilities (such as PIP).

3.2 ABORT OR STOP

Force an orderly end to a task that is running.

Format

ABORT taskname

or

STOP taskname

Where:

taskname	is the name of the task to be aborted.

Description

Flow first connects to the specified task, making Flow the task's parent. (Aborting a parentless task crashes the system.)

P/OS Directive

ABRT$

Example

```
> RUN/TASK/GO mytask
> ABORT mytask

> ABORT TMAIL
```

3.3 ACTIVATE

Instruct P/OS to activate a task (not a spawn).

Format

ACTIVATE instname

Where:

instname	is the installed name of the task you are activating.

Description

P/OS activates and subsequently runs the specified task contingent upon priority and memory availability. The ACTIVATE function is the basic mechanism that running tasks use to initiate other installed (dormant) tasks. ACTIVATE does not attempt to install the specified task if it is not installed.

P/OS Directive

RQST$ -- Request Task

Example

```
> ACTIVATE TMAIL   ! Wake up TMAIL to transfer mail
```

3.4 ASSIGN

Associate a logical name with a physical device name, a complete file specification, or another logical name.

Format

```
ASSIGN  equivalence logname
```

Where:

equivalence	is a string to which the new logical name refers. Its maximum length is 512 bytes.
logname	is the name of the logical that you are creating. Its maximum length is 30 bytes.

Description

The difference between a symbol and a logical is that Flow stores symbols in a symbol table that resides on disk. Consequently, symbols are always available, even if you terminate PRO/Office Workstation and later re-execute it. Logicals, however, are handled by P/OS, which only temporarily stores them in main memory. P/OS does not preserve logicals after terminating the task in which you define them.

You can display a current logical assignment by invoking the SHOW LOGICAL function. Also, you can delete a current logical by invoking the DELETE/LOGICAL function.

See the DEFINE/LOGICAL function, which performs the same operation in a different format.

P/OS System Routine

PROLOG

Examples

```
> ASSIGN "DW1:[USERFILES]TEST1.TSK"   TEST1
> RUN/TASK  TEST1
> DELETE/LOGICAL  TEST1
```

3.5 BLOCK

Stop the issuing task.

Format

BLOCK

Description

To unblock a task that you stop using BLOCK, you must invoke the UNBLOCK function from another task.

P/OS Directive

STOP$S

Example

```
> RUN/TASK/GO  other_task
>  .
>  .    ! You continue processing.
>  .
> BLOCK ! Block yourself.
                ! <---- Later, other_task unblocks you.
```

3.6 BLOCK/SUSPEND

Suspend the issuing task.

Format

BLOCK/SUSPEND

Description

A task can suspend only itself, not another task. You can restart the suspended task by invoking UNBLOCK/SUSPEND.

P/OS Directive

SPND$S

Example

```
> RUN/TASK/GO  other_task
>  .
>  .            ! You continue processing.
>  .
> BLOCK/SUSPEND ! Suspend yourself.
                        ! <---- Later, other_task resumes you.
```

3.7 CALL

Invoke a facility via the Software Bus.

Format

CALL SWB$name[/qualifier...]

Where:

SWB$name	is the installed name of the task you want to call. You can only CALL a PRO/Office Workstation task whose installed name begins with SWB$.

Description

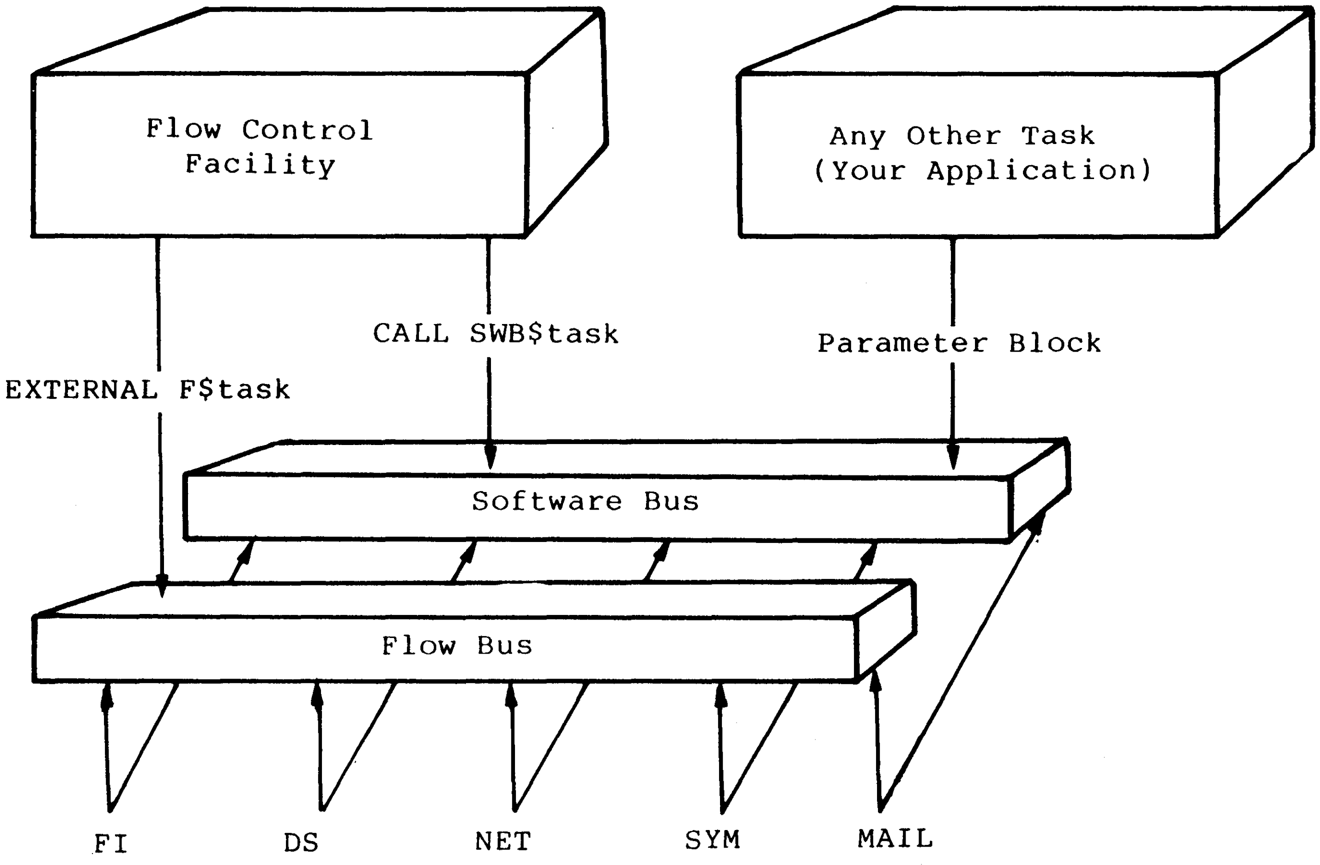
For most facilities in PRO/Office Workstation, you can access only a subset of the facility's total capability via such Flow functions as MENU, DOC, MAIL, FIELD, and FORM. To access a facility's total capability, you must invoke it via the Software Bus by using the CALL function.

For example, you can invoke Callable Document Services (CDS) directly, without having to use the Document Services Interface (DSI) as an intermediary. This allows you greater control of the document database than DSI provides.

Note that the Flow functions such as MAIL, FORM, and FIELD, translate into an invocation of the EXTERNAL function. A simple guideline to use in determining whether you should specify either the CALL or EXTERNAL function is to inspect the installed name of the task you want to invoke. If the task name begins with the prefix F$, then you use EXTERNAL (or one of the functions that translates into EXTERNAL); if the task name begins with the prefix SWB$, then you use CALL. The Pro/Tool Kit SHOW TASKS command allows you to see the names of installed tasks.

The reason for the naming convention of F$ and SWB$ involves the means by which Flow communicates with the other facilities. Flow uses two data structures to communicate with the other facilities: the Flow Bus and the Software Bus. You access modules that have installed task names beginning with F$ via the Flow Bus from Flow. On the other hand, you access modules with names beginning with SWB$ from any task (including Flow) via the Software Bus.

Figure 3-1 illustrates the Flow Bus and the Software Bus. The figure shows that Flow can communicate with the other facilities by invoking EXTERNAL to access the Flow Bus, or by invoking CALL to access the Software Bus. Any other task (an application that you build, for example), can also communicate with the other facilities by passing a parameter block on the Software Bus. (Note that making calls from applications directly over the Software bus is not supported in Version 1.0 of PRO/Office Workstation.)



FI = Form Interface Facility
DS = Document Services Facility
NET= Network Services Facility
SYMS = Symbol Services Facility
MAIL = Mail Services Facility

Figure 3-1: The Flow Bus and the Software Bus

The qualifiers are:

/PREFIX=string

If the call is successful and you specified the /PREFIX qualifier, Flow defines all parameters the called facility returns as symbols in the process table. The symbol names have the format

string+parameter_name

Where:

string	is the value you specify in the /PREFIX qualifier.
parameter_name	is the actual name of the parameter(s) returned by the module called.

The plus sign (+) indicates concatenation of the string and parameter_name.

Use /PREFIX to save the data returned from a module in a "set" of parameters chosen by the caller. You can specify the /DISPLAY qualifier to show all returned parameters on the terminal screen/

/DISPLAY

This qualifier displays returned parameters on the terminal screen.

Examples

```
> CALL SWB$DS !  invoke CDS
> CALL SWB$SB !  invoke Symbol Table Facility
> CALL SWB$FI !  invoke the Form Interface Facility

> CALL SWB$DS/$REQ=GET//$NUM=52/PREFIX="CDS."
> WRITE SYS$OUTPUT "The author is 'CDS.AUT'."
```

3.8 CLEAR

Erase the terminal screen.

Format

CLEAR

Description

Invoking CLEAR is equivalent to writing the escape sequence <ESC> [2 J (erase in display) to the terminal screen.

Example

```
> CLEAR
```

3.9 COMMAND OR @

Cause Flow to accept commands either from SYS$INPUT or from a command procedure.

Format

COMMAND [filespec]

or

@[filespec]

Where:

filespec	is the file specification of a command procedure.

Description

If you specify the name of a command procedure, Flow accepts commands from the file identified by filespec. In this case, using COMMAND is equivalent to using the at sign (@).

If you do not specify the name of a command procedure, Flow accepts commands from SYS$INPUT.

Examples

```
> COMMAND DW1:[COMMANDS]COMPRC.COM
> COMMAND
```

3.10 CONTINUE

Cause Flow's execution to proceed.

Format

```
CONTINUE
```

Description

CONTINUE is appropriate in an error trap that you want Flow to ignore, as follows:

```
> ON ERROR THEN CONTINUE
```

3.11 COPY

Create a new file from one or more existing files.

Format

```
COPY inputspec outputspec
```

inputspec	Specifies the input file to be copied.
outputspec	Specifies the output file to which the input file is copied.

Description

You can change the name, type, and version number of the file when you enter the outputspec parameter. Wildcards in the place of the name and the type leave the name and type unchanged. If you use a wildcard in either of the parameters, you must use a wildcard in both.

COPY always creates the output file. For example, if you type:

```
> COPY FILE1.LIS FILE2.LIS
```

and FILE2 already exists, COPY will create a new version of the file one higher than the existing version. If FILE2 does not already exist, COPY will create a file with the name FILE2 and extension .LIS. If you specify a version number for the output file field, then a file of that version number is created. If such a file already exists, the operation fails.

Wildcards are acceptable for output files if the destination is another directory.

You can send copies to devices as well as to directories.

You can also use the COPY command to create multiple copies of the same file with the same or different names.

Examples

```
> COPY MYFILE.DAT *.*
> COPY MYFILE.DAT DW1:
> COPY MYFILE.DAT DW1:[USERFILES]
> COPY MYFILE.DAT [USERFILES]
```

3.12 CREATE/DIRECTORY

Create a new directory.

Format

```
CREATE/DIRECTORY [dev:]dirspec
```

Where:

dev	is an optional device name, such as DW1: or DZ1:.
dirspec	is a directory name enclosed in square brackets.

Description

This function crates a directory ona disk device. An error occurs if the directory already exists.

P/OS System Routine

PRODIR

Examples

```
> CRE/DIR DW1:[MYDIR]
```

3.13 DEFINE/APPLICATION

Define a Flow symbol whose equivalence string is the name of an application installed on P/OS.

Format

```
DEFINE/APPLICATION[/qualifier...] appname
```

Where:

appname	is the symbol that you want to represent an application installed on the P/OS Main Menu. All rules regarding symbols apply to appname. See the description of SYMBOL/DEFINE for details.

Description

Flow does the following:

- o Displays all applications installed on P/OS.
- o Allows you to select the application that appname will represent.

In order to invoke a P/OS-installed application, you must first use DEFINE/APPLICATION to make it known to Flow.

Once you have defined an application, you can invoke by using the run command:

```
> RUN/APPLICATION appname
```

See Chapter 6 for a description of the following qualifiers that you can specify for this function:

/NODELETE
/OVERRIDE
/PROCESS
/SYSTEM
/USER
/VOLATILE

Example

```
> DEF/APPL  SUPERCOMP
```

3.14 DEFINE/COMMAND

Create a dynamic command.

Format

```
DEFINE/COMMAND[/qualifier...] dyncommand equivalence
```

Where:

dyncommand	is the name of the dynamic command you are defining. Note that this name is merely a special kind of symbol.
equivalence	is any static command (function) listed in this chapter.

Description

This function allows you to define your own commands, called <u>dynamic commands</u>. Each dynamic command must resolve to a Flow function, also called a <u>static command</u>.

Examples

The following example defines the dynamic command PIP. The static command equivalence runs the Tool Kit PIP utility. Note that the equivalence string uses the /CMD qualifier to automatically pass parameters to PIP. Also note the double quotes to distinguish the first delimiter of the /CMD qualifier from the last delimiter of the DEFINE/COMMAND function.

```
> DEFINE/COMMAND PIP  "RUN/TASK  PIP/CMD=""PIP "
> PIP X/LI
```

Flow translates the command PIP X/LI as:

```
> RUN/TASK PIP/CMD="PIP X/LI"
```

NOTE

Flow functions have precedence over dynamic functions.

See Chapter 6 for a description of the following qualifiers that you can specify for this function:

/NODELETE
/OVERRIDE
/PROCESS
/SYSTEM
/USER
/VOLATILE

3.15 DEFINE/KEY

Assign a Flow function to a specified key.

Format

DEFINE/KEY[/qualifier...] keynumber equivalence

Where:

keynumber	is the numeric value of the key that you are defining.
equivalence	is any Flow function or command procedure invocation.

Description

This command lets you store a Flow function in a function key or keypad key on the Professional's terminal keyboard. Once you have defined a key, pressing it causes Flow to execute the stored command(s).

Note that the Dumb Terminal Emulator (DTE) disables key definitions that you have established with DEFINE/KEY.

Table 3-1 lists the numeric value of each function and keypad key on the Professional 350 keyboard. You cannot define keys that the table lists as reserved.

Table 3-1: Key Values

Key Type	Key Value	Key Name
Function Keys	1	Reserved
	2	Reserved
	3	BREAK
	4	SETUP
	5	F5
	6	Reserved
	7	RESUME
	8	CANCEL

Key Type	Key Value	Key Name
	9	MAIN SCREEN
	10	EXIT
	11	F11
	12	F12
	13	F13
	14	ADDTNL OPTIONS
	15	HELP
	16	DO
	17	F17
	18	F18
	19	F19
	20	F20
	21	FIND
	22	INSERT HERE
	23	REMOVE
	24	SELECT
	25	PREV SCREEN
	26	NEXT SCREEN
	27	up arrow
	28	left arrow
	29	down arrow
	30	right arrow
	31	PF1
	32	PF2

Key Type	Key Value	Key Name
	33	PF3
	34	PF4
Keypad Keys	35	minus
	36	comma
	37	period
	38	ENTER
	39	0
	40	1
	41	2
	42	3
	43	4
	44	5
	45	6
	46	7
	47	8
	48	9

See Chapter 6 for a description of the following qualifiers that you can specify for this function:

/NODELETE
/OVERRIDE
/PROCESS
/SYSTEM
/USER
/VOLATILE

Examples

```
> DEF/KEY 19 "DIR"
> DEF/KEY 20 "RUN DTE"
```

3.16 DEFINE/LOGICAL

Associate a logical name to a physical device name, a complete file specification, or another logical name.

Format

DEFINE/LOGICAL logname equivalence

Where:

logname	is the name of the logical that you are creating. Its maximum length is 30 bytes.
equivalence	is a string to which the new logical name refers. Its maximum length is 512 bytes.

Description

The difference between a symbol and a logical is that Flow stores symbols in a symbol table that resides on disk. Consequently, symbols are always available, even if you terminate PRO/Office Workstation and later re-execute it. Logicals, however, are handled by P/OS, which only temporarily stores them in main memory. P/OS does not preserve logicals after terminating the task in which you define them.

You can display a current logical assignment by invoking the SHOW LOGICAL function. Also, you can delete a current logical by invoking the DELETE/LOGICAL function.

See also the ASSIGN command, which performs the same operation in a different format.

P/OS System Routine

PROLOG

Examples

```
> DEFINE/LOG TEST1 "DW1:[USERFILES]TEST1.TSK"
> RUN/TASK  TEST1
> DELETE/LOGICAL  TEST1
```

3.17 DEFINE/SYMBOL

Create a symbol table entry and assign an equivalence name to it.

Format

```
DEFINE/SYMBOL[/qualifier...] symname  equivalence
```

Where:

symname	is the name of the symbol that you are creating. Its maximum length is 30 characters.
equivalence	is a string to which the new symbol refers. Its maximum length is 512 bytes.

Description

Flow strips spaces, tabs, and control characters from the symbol name and the equivalence string. Additionally, Flow treats the slash (/) in a equivalence string as a delimiter. To retain any of these characters in the equivalence string, you must surround the string with quotation marks.

Note you can specify quotation marks anywhere in the command line. To represent quotation marks within a quoted equivalence string, you must double the quotation marks within the string.

Also, note that Flow changes the equivalence string to all uppercase characters. Again, you can surround the string with quotation marks to override this.

You should not define the symbols that begin with the following characters (they are reserved):

- FLOW$
- FI$
- OA$
- MAIL$
- XNET$
- DTF$

- ORDA$

The difference between a symbol and a logical is that Flow stores symbols in a symbol table that resides on disk. Consequently, symbols are always available, even if you terminate PRO/Office Workstation and later re-execute it. Logicals, however, are handled by P/OS, which only temporarily stores them in main memory. P/OS does not preserve logicals after terminating the task in which you define them.

See Chapter 6 for a description of the following qualifiers that you can specify for this function:

/NODELETE
/OVERRIDE
/PROCESS
/SYSTEM
/USER
/VOLATILE

Examples

```
> DEF/SYM/SYS  FLOW$_STARTUP_FUNCTION  "@DW1:[COMMANDS]LOGIN.COM"
> DEF/SYM  MYNAME  "MARTY FRIEDMAN"
```

3.18 DEFINE/TAG

Define a tag for a Form Interface form.

Format

DEFINE/TAG[/qualifier...] tagname equivalence

Where:

tagname	is the name of the tag you are defining. Do not include the dollar sign ($) in a tag; this symbol delimits a tag from a form name in a complete form specification. The maximum length of the tagname is 30 bytes.
equivalence	is a series of qualifiers that provide information to FI about the tag. The maximum length of the equivalence is 512 bytes.

Description

See Table 4-1 in Chapter 4 for a desciption of the qualifiers you can specify in the equivalence.

See Chapter 6 for a description of the following qualifiers that you can specify on the DEFINE/TAG function:

/NODELETE
/OVERRIDE
/PROCESS
/SYSTEM
/USER
/VOLATILE

Example

```
> DEFINE/TAG MYDFLT "/LIB=MYLIB.FLB/FORM=MYMAIN/DEF=A1/DIS=FMS"
> FORM MYDFLT            !use tag only

> DEFINE/TAG MYDFLT "/LIB=MYLIB.FLB/DEF=A1/DIS=FMS"
> FORM MYDFLT$MYMAIN     !use tag in form specification
```

3.19 DELETE/APPLICATION

Delete a Flow symbol representing an application installed on the P/OS Main Menu.

Format

```
DELETE/APPLICATION[/qualifier...] appname
```

Where:

appname	is the symbol that you want to delete, having previously defined it using DEFINE/APPLICATION.

Description

You can look at DEFINE/APPLICATION and DELETE/APPLICATION as special cases of DEFINE/SYMBOL and DELETE/SYMBOL. Here we refer to special symbols that represent P/OS applications.

See Chapter 6 for information on the following qualifiers that you can specify:

/OVERRIDE
/PROCESS
/SYSTEM
/USER

Example

```
> DEL/APPL/SYS  SUPERCOMP
```

3.20 DELETE/COMMAND

Delete a dynamic command definition.

Format

```
DELETE/COMMAND[/qualifier...] dyncommand
```

Where:

dyncommand	is the symbol that you want to delete, having previously defined it using DEFINE/COMMAND.

Description

Once you have deleted a dynamic command definition, you can no longer refer to the dynamic command name.

See Chapter 6 for information on the following qualifiers that you can specify:

/OVERRIDE
/PROCESS
/SYSTEM
/USER

Example

```
> DEL/COM/SYS  SPAWN
```

3.21 DELETE/DIRECTORY

Delete a directory.

Format

DELETE/DIRECTORY [dev:]dirspec

Where:

dev:	is an optional device name, such as DW1: or DZ1:.
dirspec	is a directory name enclosed in square brackets.

Description

This function deletes a directory from your system.

P/OS System Routine

PRODIR

Example

```
> DEL/DIR  DW1:[MYDIR]
```

3.22 DELETE/KEY

Remove a key definition established with DEFINE/KEY.

Format

```
DELETE/KEY[/qualifier...] keynumber
```

Where:

keynumber	is the keypad or function key value of the key that you have previously defined with DEFINE/KEY. See the description of DEFINE/KEY for details.

Description

Use DEFINE/KEY to redefine a key whose definition you have deleted.

See Chapter 6 for information on the following qualifiers that you can specify:

/OVERRIDE
/PROCESS
/SYSTEM
/USER

Example

```
> DELETE/KEY/PROC  19
```

3.23 DELETE/LOGICAL

Delete a logical name definition.

Format

```
DELETE/LOGICAL logname
```

Where:

logname	is the name of the logical to be deleted.

Description

Use the DEFINE/LOGICAL command to redefine a logical that you have deleted.

P/OS System Routine

PROLOG

Example

```
> DEL/LOG  MYLOG
```

3.24 DELETE/SYMBOL

Delete a symbol definition from the symbol table.

Format

```
DELETE/SYMBOL[/qualifier...] symname
```

Where:

symname	Is the name of the symbol to be deleted. Its maximum length is 30 characters.

Description

Use the DEFINE/SYMBOL command to redefine a symbol that you have deleted.

See Chapter 6 for information on the following qualifiers that you can specify:

/OVERRIDE
/PROCESS
/SYSTEM
/USER

Example

```
> DEL/SYM/SYS  MYSYM
```

3.25 DELETE/TAG

Remove a a tag name for a Form Interface form.

Format

```
DELETE/TAG[/qualifier...] tagname
```

Where:

tagname	is the tag name of an FI form.

Description

See Chapter 4 for details on tags.

See Chapter 6 for information on the following qualifiers that you can specify:

/OVERRIDE
/PROCESS
/SYSTEM
/USER

Example

```
> DEL/TAG/SYS  MY_TAG
```

3.26 DIRECTORY OR SHOW DIRECTORY

Display information for an individual file or a group of files.

Format

```
DIRECTORY [filespec [,filespec]...]
```

or

```
SHOW DIRECTORY [filespec [,filespec]...]
```

Where:

filespec	is the file specification for the file for which you want information. You can specify none, one, or several file specifications.

Description

Specifies the file or files for which information should be displayed. If you do not supply a filespec, a complete directory for the default directory is displayed.

You can supply one or more filespecs, separated by commas. If you do not supply a version number, only information on the most recent versions is displayed.

You can use a wildcard in any field except the device field. The default value of the filespec is *.*. See the examples below.

You can display another directory by supplying the directory name in this field. You can also specify device names in the form ddnn: in this field.

Examples

Since the default value of the filespec is *.*., the following commands are equivalent (assuming that your current device and directory are DW1:[USERFILES]):

```
> DIR *.*
> DIR
> DIR DW1:
> DIR DW1:[USERFILES]
> DIR [USERFILES]
```

3.27 DISMOUNT

Declare a volume to be logically dismounted.

Format

```
DISMOUNT dev:
```

Where:

dev:	is the device on which the volume is currently mounted.

Description

This function allows you to manually dismount a volume.

P/OS System Routine

PROVOL

Examples

```
> DISMOUNT DZ1:
> DISMOUNT OA:
```

3.28 DOC

Invoke a Document Services Facility function.

Format

DOC dsfunction

Where:

dsfunction	is one of the Document Services functions described in Chapter 5.

Description

This function allows you to manipulate the document database via the Document Services Interface (DSI). For further information, see Chapter 5.

Examples

```
> DOC CAB/REQ=SELECT    ! Select a file cabinet
> DOC DISPLAY           ! Display the current document
> DOC EDIT/$NUM=34     ! Edit document 34 in current cabinet
```

3.29 EDT

Run the EDT editor.

Format

```
EDT [inputspec]
```

Where:

inputspec	is the file specification of the file that you want to edit. If the file does not exist, P/OS creates it for you. If you do not specify inputspec, EDT returns the EDT> prompt. This allows you to enter commands in MCR format.

Description

Flow installs EDT when you start up the PRO/Office Workstation application.

The PRO/Office Workstation kit contains two EDT startup command files:

- [1,2]EDTSYS.EDT
- 'FLOW$_EXE_ACCOUNT'EDTINI.EDT.

When you invoke EDT, it always reads [1,2]EDTSYS.EDT first. This file contains definitions for various keys, including editing keypad keys, the HELP and DO keys, and some of the function keys on the top row. Also, the file defines the FIND, INSERT HERE, REMOVE, SELECT, PREVIOUS SCREEN, and NEXT SCREEN keys.

You can edit [1,2]EDTSYS.EDT, placing the following line at the end of the file:

```
SET COMMAND EDTINI
```

This causes EDT to look in the current directory for a file called EDTINI.EDT, which can contain more key definitions and startup commands.

EDT supports the full DEC Multinational Character Set generated by the Professional keyboard, including characters generated by compose sequences. Also, EDT uses the MCR command line format for specifying input and output files. See the EDT documentation for a description of that command format.

3.30 EMIT

Write a value to the symbol $RESULT.

Format

EMIT[/qualifier] integer

Where:

integer	is a decimal, integer value.

Description

EMIT is a means of sending status information from an offspring task to a parent task. The function allows you to synchronize a parent and offspring task.

The qualifier is:

/NAME=taskname

Where:

taskname	is the name of the task to which you wish to emit information. The task must be a parent of the emitting task (that is, it must have spawned or be connected to the emitting task).

Example

Suppose you write a command procedure called EMITST.COM, as follows:

```
> SET VERIFY
> ON ERROR THEN CONTINUE
> WRITE SYS$OUTPUT "BEGINNING EMITST.COM"
> WAIT 2
> WRITE SYS$OUTPUT "EMITTING STATUS"
> EMIT/NAME=FLOW 6
> WAIT 2
> WRITE SYS$OUTPUT "TERMINATING EMITST.COM"
> EXIT
```

Then, suppose you invoke this command procedure from Flow CLI, as follows:

```
RUN/TASK/GO/NAME=SPAWN [ZZFLOW]FLOW.TSK/CMD="@[COMMANDS]EMITST"
```

(Assume that the command procedure is in directory [COMMANDS].)

That CLI command creates an offspring Flow task (named SPAWN), which executes the command procedure. The command procedure emits the value 6 to the parent Flow task. The symbol $RESULT in the parent process table will contain the value "6". Note that the parent and the offspring continue running after the emit. You can use this function to perform parallel processing.

3.31 END

Terminate the current Flow task.

Format

```
END
```

Description

END is useful when you have invoked more than one Flow task, and you want to terminate the current Flow task. Note that you must invoke END to terminate the first Flow task (the one installed by the installation file during PRO/Office Workstation startup).

Example

Suppose a command procedure contains the statement:

```
> ON ERROR THEN END
```

If you execute the command procedure from an offspring Flow and the ERROR condition arises, the offspring Flow terminates. You would return to the parent Flow.

Note, however, that the END statement in a Flow task that you spawned from a parent Flow is not required. An offspring Flow task automatically terminates when the command file it is executing completes.

3.32 EXIT

Leave the current environment.

Format

```
EXIT
```

Description

Invoking EXIT causes you to pop the current Flow command from the Flow stack. This brings you back to the Flow command just prior to the one last executed.

Note that when the Flow stack contains only one command -- the first command -- EXIT will re-execute that command. Consequently, you cannot use EXIT to terminate the first Flow function (either MENU MAIN, the command eqivalence of FLOW$_FIRST_FUNCTION, or the menu selected from the SUFIR menu). To terminate the first Flow function, use the END command.

Example

```
> EXIT
```

3.33 EXTERNAL

Invoke a facility via the Flow Bus.

Format

```
EXTERNAL[/qualifier] F$name [servcom]
```

Where:

F$name	is the installed name of the task you want to invoke. Currently, these can be: 1. F$DS (invokes DSI - Document Services Interface) 2. F$MAIL (invokes Flow Mail Services)
function	is a facility function that you can pass to the facility upon activating it.

Description

The DOC and MAIL functions both expand to the EXTERNAL function specifying the appropriate task name. Use DOC and MAIL whenever possible.

See Section 3.7 for a description of the Flow Bus, as well as a description of the difference between the CALL and EXTERNAL functions.

The qualifier is:

/INSTALL=filename.TSK

This qualifier installs, runs, and removes filename.TSK just before executing F$name.

Example

The following function invokes the Document Services Interface and passes it the Document Services command CREATE. This is equivalent to using the function DOC CREATE.

```
> EXTERNAL F$DSI CREATE/.TIT="This is the title."
```

3.34 FIELD

Control the input and output of field data on a form.

Format

```
FIELD[/qualifier...] formspec  fieldname[/qualifier]
```

Where:

formspec	is the form specification of the form containing the field that the Form Interface Facility activates.
fieldname	is the FMS name of an enterable field in the specified form.

Description

Calling the FIELD function is equivalent to calling the FORM function and specifying the /CHOICE qualifier to activate a selected field.

A form specification has the format:

tag$formname

Where:

tag — is a tag you have defined with DEFINE/TAG. For example:

```
> DEFINE/TAG MYTAG "/LIB=LIBR.FLB/DEF=A1"
```

formname — is the name of the desired form.

After defining the tag MYLIBR, you could invoke the FIELD command as follows:

```
> FIELD MYTAG$MYFORM CHOICE/OUT="DATA_TO_DISPLAY"
```

The Form Interface Facility searches in the library LIBR.FLB for the form called MYFORM.FRM, which contains a field ACHOICE. The user can enter a choice in this field.

If you do not specify a tag, Flow translates the default tag, DEFAULT. This tag contains the name of the default library, OAFMS.FLB. For example, if you place MYFORM in the default library, you could invoke the FIELD command without using a tag:

```
> FIELD MYFORM ACHOICE
```

On the other hand, you can specify a library and a form in the tag specification:

```
> DEFINE/TAG MYTAG  "/LIB=OAFMS.FLB/FOR=MYFORM.FRM/DEF=A1"
```

Having defined such a tag, you could then invoke the FIELD function without specifying a formname:

```
> FIELD MYTAG       !specify a tag only
```

See Chapter 4 for details regarding tags.

The qualifiers are:

/SYMBOL=symbolname

Specifies either of the following:

- a symbol into which the Form Interface places the retrieved input if you specified /INPUT,
- a symbol from which the Form Interface reads the outputtext if you also specify the parameter qualifier /OUTPUT.

/CLEAR

Clears the screen and draws the current form.

There are parameter qualifiers for the fieldname parameter:

/OUTPUT[=outputtext]

Causes the Form Interface to put outputtext in the specified field. You must specify the /SYMBOL=symbolname qualifier if you omit =outputtext from /OUTPUT.

/INPUT

Allows the Form Interface to get input from the specified field.

3.35 FIX

Load and locks an installed task or region in memory.

Format

FIX[/qualifier] instname

Where:

instname	is the installed name of the task that you want to fix.

Description

Once a task is fixed in a memory partition, P/OS can service subsequent requests for the task much more quickly than if the task were not fixed. This is because a fixed task is memory-resident and P/OS does not have to load it from disk.

You cannot fix an active task. However, you must install a task before fixing it. Also, fixed tasks remain memory-resident even after they exit or abort.

Not all tasks run properly when fixed. A task might require data areas to contain certain values when loaded in from the disk. The first time the task is run, these data areas might be modified and the task can run unpredictably thereafter. Tasks that initialize their data areas, and therefore do not have this problem, are called serially reentrant.

You can fix an overlaid task. If its root segments are serially reentrant, it will run correctly. However, since P/OS must still read the tasks's overlaid segments from disk (unless you built the task using memory-resident overlays), you gain little by fixing it.

The following qualifier is available:

/REGION

Specifies that you want to fix a common region rather than a task.

P/OS System Routine

PROTSK

Examples

```
> FIX test.tsk
> FIX
```

3.36 FORM

Invoke the Form Interface Facility, which displays the specified form.

Format

```
FORM[/qualifier...] formname
```

Where:

formname	is the name of a form as it appears either in an FMS or FDT library.

Description

The FORM command invokes the Form Interface (FI), causing it to display a form. FI opens the library containing the form, scans the named data for form specifications (in the case of FMS), displays the form, and then waits for input from any specified fields.

You must specify a form; there is no default form. If you do not specify a field (by specifying /CHOICE), then FI uses the default field CHOICE.

The qualifiers are:

/FORM=formname

Specifies the name of the desired form (formname) within an FMS or FDT library.

/LIBRARY=libname

Specifies the name of the FMS or FDT library (libname) from which FI is to extract the form.

/FILE=filespec

Specifies a file specification containing form text for a static form. See the chapter on the Form Interface Facility for a description of static forms.

/DEFINITION

You can also specify this qualifier in a tag definition. See Chapter 4 for a description of this qualifier.

/DISPLAY

You can also specify this qualifier in a tag definition. See Chapter 4 for a description of this qualifier.

Examples

```
> FORM OA$PROFILE
> FORM/DEF=A1/DISP=FMS/LIB=OAFMS.FLB/FORM=PROFIL
```

3.37 GOTO

Transfer execution to a labeled statement in a command procedure.

Format

GOTO label

Where:

label	specifies an alphanumeric label appearing as the first item (after the dollar sign) in a command line. When Flow executes the GOTO command, execution passes to the command following the specified label.

Description

The label must follow the GOTO statement in the current command procedure. It must be terminated with a colon (:) and cannot contain blanks. If the specified label does not exist, the procedure exits.

Example

```
$ GOTO mylabel
```

3.38 HELP

Display help.

Format

```
HELP
```

Description

This function displays a frame that provides help on the CLI commands.

Example

```
> HELP
```

3.39 IF ... THEN

Conditionally execute a command based on the value of an expression.

Format

IF expression THEN command

Where:

expression	is the test that Flow performs.
command	is the function that Flow executes if the value of the expression is true.

Description

Flow's expression evaluation consists of general logical comparisons and a limited number of arithmetic functions. The expression <u>must</u> resolve to a boolean value. Note that Flow does not resolve all arithmetic expressions to a boolean value. Arithmetic operations support integers only.

Note that parenthetical ordering is not supported; all operations occur left to right.

Flow translates strings into all uppercase characters for comparisons. Strings beginning with "Y" or "T" are logically true; all others are false.

The specific operations you can use in an expression are

String logical operations:

.eqs.	- equal string
.nes.	- not equal string
.ges.	- greater than or equal string
.gts.	- greater than string
.les.	- less than or equal string
.lts.	- less than string

Integer logical operations:

.eq.	- equal to
.ne.	- not equal to
.ge.	- greater than or equal to
.gt.	- greater than
.le.	- less than or equal to
.lt.	- less than

.and. - logical AND of integer bits
.or. - logical OR of integer bits
.xor. - logical exclusive OR of integer bits

Integer arithmetic operations:

+ - add integers
- - subtract integers
* - multiple integers
/ - divide integers
.not. - complement of an integer

You can use IF to check the FLOW status from a command ($STATUS) or the task exit status from a RUN/TASK command ($RESULT). You can also use IF in conjunction with the LET command to build loops with counters.

Note that any qualifiers you specify on this function must appear after the THEN keyword, not the IF keyword. (Only the global qualifiers described in Section 3.1 are available.)

Examples

```
$ IF a .eqs. b THEN/WARNING/QUIET CONTINUE
$ IF $STATUS .and. 1 THEN WRITE SYS$OUTPUT "Success!!!"
$ IF a .eqs. b THEN GOTO aequalsb
$ IF counter .lt. 10 THEN GOTO loop
$ IF a*10 .eq. 500 THEN EXIT
```

The following are examples of ILLEGAL syntax:

```
$ !mathematic, not logical expression:
$ IF a*10 THEN EXIT

$ !Correct syntax is:
$ IF a*10 .and.  1 THEN EXIT

$ !parentheses are not supported:
$ IF (a.eqs.b) .and.  (c.ne.d) THEN GOTO label

$ !Correct syntax is:
$ IF a.nes.b THEN GOTO nottrue
$ IF c.eq.d THEN GOTO nottrue
```

3.40 INQUIRE

Interactively assign a value to a symbol during execution of a command procedure.

Format

INQUIRE symbol [promptstr]

Where:

symbol	is the name of a symbol. It can be a symbol that you have previously defined with SYMBOL/DEFINE, or it can be a symbol that the INQUIRE function defines for you (in the symbol process table).
promptstr	is the prompt string that Flow displays on the terminal screen upon executing the INQUIRE command. If the prompt string contains any lowercase characters, multiple blanks, or tabs, or at sign (@) characters, enclose the entire string in quotation marks ("promptstr").

Description

This function allows a procedure to display a prompt and assign a string to a symbol. Note this is analogous to the MENU/RETURN and FIELD commands, which use the Form Interface Facility to perform the same operation.

The qualifier is:

/[NO]PUNCTUATION

Controls whether or not a colon (:) and a space follow the prompt when Flow displays it on the terminal screen. By default, Flow provides this punctuation. If you want to suppress the colon and space, speckfy /NOPUNCTUATION.

See Chapter 6 for information on the other qualifiers that you can specify:

/NODELETE
/OVERRIDE
/PROCESS
/SYSTEM
/USER
/VOLATILE

Example

```
> DEFINE/SYMBOL CONT  N
> INQUIRE CONT "Enter Y to continue: "
> IF .NOT. CONT THEN EXIT
```

3.41 INSTALL

Include a task image, common block, or library in the System Task Directory, thus making it known to the system.

Format

```
INSTALL[/qualifier...] filespec
```

Where:

filespec	is the file specification of the task image, common block, or library that you want to install. The default type is .TSK.

Description

Flow stores the installed name of a task, library, or region in the symbol $NAME. Note that the RUN/APPLICATION function can also install tasks, libraries, and regions. The installed name of an image often differs from that of the file specification name.

The symbol $NAME is actually defined through the Symbol Table Facility, not as a special symbol in Flow. It is defined as a volatile symbol in the process table.

An installed task is dormant until the P/OS executive requests it to run. You can request an installed task to run by invoking the RUN/TASK function.

Normally, if you attempt to install a task (or common or library) with a name that is already in use, you raise the fatal ERROR condition, causing an executing command procedure to abort. However, by specifying the /WARNING qualifier you convert the ERROR condition to a nonfatal WARNING condition. Note that this conversion occurs only when the ERROR condition arises as a result of your attempt to specify a name that is already in use.

The qualifiers are:

/NAME=instname

> Specifies the name (instname) by which you can later refer to the installed task. The Professional Application Builder (PAB) sets this default during linking. The /NAME qualifier overrides the PAB default.

/FIX

Causes Flow to call the P/OS callable routine PROTSK to fix the task, common block, or library in memory. See the _P/OS System Reference Manual_ for further information on PROTSK.

3.42 LET

Assign the value of an expression to a symbol.

Format

LET symbol = expression

Where:

symbol	is the name of a symbol. It can be a symbol that you have previously defined with SYMBOL/DEFINE, or it can be a symbol that the LET function defines for you (in the symbol process table).
expression	is an expression whose result you assign to the symbol.

Description

Note that you can use combinations of the IF, LET, and GOTO commands to form loops.

Examples

```
> LET a = 1+2+3+4           ! General arithmetic
> LET b = 4*a                ! Use of symbols
> LET a = "This is nice"    ! Note literal is uppercased
> LET a = $status .and. 1   ! Returns "T" or "F"
```

3.43 MAIL

Invoke a Mail Services Facility function.

Format

MAIL msfunc

Where:

msfunc	is one of the Mail Services Facility functions described in Chapter 7.

Description

This function allows you to handle mail messages. For further information, see Chapter 7.

Examples

```
> MAIL CREATE
> MAIL MORE
```

3.44 MENU

Invoke the Form Interface Facility, which displays the specified menu.

Format

```
MENU[/qualifier...] menuname
```

Where:

menuname	is the name of a menu as it appears either in an FMS or FDT library.

Description

The MENU command invokes the Form Interface (FI), causing it to display a menu. FI opens the library containing the form, scans the named data for form specifications (in the case of FMS), displays the form, and then waits for input from any specified fields.

You must specify a form; there is no default form. If you do not specify a field (by specifying /CHOICE), then FI uses the default field CHOICE.

The qualifiers are:

/[NO]ALLOW

IF you specify /ALLOW, the Form Interface Facility lets Flow attempt to process the user's choice if that choice does not exist on the displayed menu. If you specify /NOALLOW, Flow cannot process the choice, and no action occurs for nonexistent choices.

/[NO]EXACT

If you specify /EXACT, the Form Interface Facility reads the entire name of a menu selection as it is stored in its form library. If the user's choice is not exactly the same as the stored name, then no match occurs. If you specify /NOEXACT, a match occurs when the user's choice coincides with the first n letters of the stored choice, where n is the length of the user's choice.

/[NO]KEY

This qualifier enables or disables recognition of escape sequences from the terminal keypad during display of the menu.

/ONCE

Displays the requested menu, allows the user to make a selection, processes the user's selection, but does not return to the requested menu after processing the selection.

The following example contrasts MENU and MENU/ONCE.

- Column 1 shows three menus, each invoked with the MENU command. Upon unwinding the Flow stack (with the EXIT or UNWIND function), FI returns each time to the previous MENU invocation. The order of display is A-B-C; the order of unwinding is C-B-A.

- Column 2 shows how MENU/ONCE causes FI to display MENU B only once, skipping that menu when unwinding. The order of display is A-B-C; the order of unwinding is C-A.

```
   Column 1           Column 2
   --------           --------

    MENU A            +-->MENU A
     ^ |              |      |
     | v              |      v
    MENU B            ^   MENU/ONCE B
     ^ |              |      |
     | v              |      v
    MENU C            +--+MENU C
```

/CHOICE

Specifies the name of a field in the form that will accept user input.

/FORM=formname

Specifies the name of the desired form (formname) within an FMS or FDT library.

/LIBRARY=libname

Specifies the name of the FMS or FDT library (libname) from which FI is to extract the form.

/FILE=filespec

Specifies a file specification containing form text for a static form. See the chapter on the Form Interface Facility for a description of static forms.

/RETURN

Causes FI to return MENU operation information that Flow stores in two symbols:

$RESULT contains the Flow function (FMS named data or FDT action string) that would be invoked as a result of the user's selection.

$CHOICE contains the actual characters the user entered to make the selection.

When you specify /RETURN, FI does not execute the user's choice; the switch only allows you to obtain the values FI places in the $RESULT and $CHOICE.

You can also specify the following qualifiers:

/DEFINITION
/DISPLAY

See the section on tag qualifiers in Chapter 4 for a description of these qualifiers.

3.45 MOUNT

Specify that a volume is on line.

Format

```
MOUNT dev:labelspec[/qualifier]
```

Where:

dev:	is the device on which you want to mount the volume.
labelspec	is the volumename enclosed in brackets.

Description

You can mount FILES-11 or foreign disk volumes.

The qualifier is:

/FOREIGN

This qualifier specifies that you are mounting a foreign disk volume.

P/OS System Routine

PROVOL

Examples

To make sure that the required disk is mounted in DZ1:, invoke DISMOUNT and MOUNT as follows:

```
> DISMOUNT DZ1:
> MOUNT DZ1:mydisk
```

3.46 NETWORK

Invoke a Network Services function.

Format

```
NETWORK[/qualifier...] nsfunc
```

Where:

nsfunc is a Network Services function.

Description

The NETWORK command runs the XCOM task, part of the Network Services Facility. See Chapter 8 for details on the Network Services functions you can perform.

The qualifiers are:

/TIMEOUT=n	The value of n is the NETWORK call timeout. By default, n equals 60 seconds.
/TERMINAL	Run the Dumb Terminal Emulator (DTE) immediately after executing the Network Services command, and exit the emulator when the NETWORK command is done.

The global qualifiers described at the beginning of this chapter are also available.

Examples

```
> NET START
> NETWORK LOGIN MOSES::M_FRIEDMAN
> NETWORK FINISH
> NETWORK STOP
> NET RUN MOSES::M_FRIEDMAN/CMD="PRINT 'OA$CURMES_FILE'"
```

3.47 ON ... THEN

Establish a trap for a specified condition.

Format

ON condition THEN function

Where:

condition	can be any of the following conditions: • ERROR • WARNING
function	is any Flow function except ON.

Description

Use the ON function to establish an error trap. Execution of the command in the ON function does not occur unless the specified condition arises.

The ERROR and WARNING conditions can arise from any Flow function within a command procedure. In order to successfully establish an error trap, you must have executed the ON function prior to executing the function that raises the specified condition.

Note that any qualifiers you specify on this function must appear after the THEN keyword, not the ON keyword. (Only the global qualifiers described in Section 3.1 are available.)

Examples

```
$ ON WARNING THEN/CLEAR CONTINUE          ! qualifier position
$ ON WARNING THEN CONTINUE                ! ignore error trap
$ ON ERROR THEN EXIT                      ! terminate
$ ON ERROR THEN GOTO error_trap           ! handle error
```

3.48 PURGE

Delete all but the latest versions of files, and release the storage space that the deleted files occupied. >Flow function

Format

```
PURGE[/qualifier] filespec[,filespec]...
```

Where:

filespec	is a file specification for a file you want to purge.

Description

PURGE is useful to clean up your directories.

The qualifiers are:

/CONFIRM

This qualifier causes Flow to prompt you for confirmation that it should delete the specified file(s).

/LOG

This qualifier specifies that the deleted files be listed on your terminal screen.

3.49 REMOVE

Delete a task name from the System Task Directory or delete a region name from the Common Block Directory.

Format

```
REMOVE[/REGION] instname
```

Where:

instname	is the installed name of the task you want to remove.

Description

To remove an active task, you must first abort the task.

If a task is fixed, REMOVE first unfixes it and then removes it.

The /REGION qualifier specifies that you want to remove a region from the Common Block Directory.

P/OS System Routine

PROTSK

3.50 RENAME

Change the name, type, or version number of an existing file.

Format

RENAME oldspec newspec

oldspec	is the file specification prior to renaming.
newspec	is the desired new file specification of the file.

Description

You can use RENAME to change not only the name of a file, but the file type or version number.

The output specification (newspec) of the RENAME command uses a default filename and file extension of "*.*". Consequently, assuming that your current device and directory are DW1:[USERFILES], the following commands are equivalent:

```
> RENAME MYFILE.DAT *.*
> RENAME MYFILE.DAT DW1:
> RENAME MYFILE.DAT DW1:[USERFILES]
> RENAME MYFILE.DAT [USERFILES]
```

3.51 RUN/APPLICATION

Run an installed and defined P/OS application.

Format

```
> RUN/APPLICATION[/qualifier]   [appname]
```

Where:

appname	is the name of the application you have previously defined using the DEFINE/APPLICATION function. Do not specify appname if you specify the /SELECT qualifier.

Description

RUN/APPLICATION allows you to invoke an application installed on P/OS. Flow invokes the task F$APPL to actually read the application's installation file and run the application.

Note that you might run an application that attempts to install tasks that are already installed. For example, if you attempt to run the PRO/Tool Kit, you receive error messages saying that the system cannot install PIP.TSK and EDT.TSK. You must remove these tasks before attempting to run the application:

```
> REMOVE ...PIP
> REMOVE ...EDT
> RUN/APPLICATION toolkit
```

After running the application you must reinstall the tasks:

```
> INSTALL [ZZFLOW]PIP
> INSTALL [ZZFLOW]EDT
```

You can easily perform these operations in a command procedure.

The qualifier is:

/SELECT

This qualifier causes Flow to display all the applications installed on the system. The user can press the PREV SCREEN and NEXT SCREEN keys to scroll the form.

Examples

```
> RUN/APPL TOOLKIT
> RUN/APPLICATION/SELECT
```

3.52 RUN/TASK

Run a task, installing it beforehand and removing it afterwards if necessary.

Format

```
RUN/TASK[/qualifier...] filespec[/qualifier]
```

Where:

filespec	is the file specification of the .TSK file containing the task image you want to run.
command	is a command that you can pass to the task you are running.

Description

The RUN function allows you to execute a task. For example, you can directly run the following P/OS (V1.7) tasks:

- C$DUTL (P/OS Disk Services)
- C$FUTL (P/OS File Services)
- C$PUTL (P/OS Print Services)
- C$SUTL (P/OS Setup)
- C$VUTL (P/OS View Message/Status Services)
- DTE (Dumb Terminal Emulator, part of PRO/Communications)

Flow returns the install/run status in the symbol $STATUS. Flow returns the numeric string of the task's exit status in the symbol $RESULT. (See an exception for the /GO qualifier.)

The symbol $STATUS exists only within Flow.

The command qualifiers are:

/NAME=taskname

> This switch specifies the name of the task as installed or as you want to install it before running it. If you do not specify /NAME, then the RUN function uses the P/OS callable system routine PROTSK to determine the task name. See the P/OS System Reference Manual for details on PROTSK.

/GO

The /GO switch causes the specified task to run asynchronously, in the background. Without /GO, the RUN/TASK function installs the task synchronously (if necessary), spawns the task, and then waits until the task terminates or emits status (see the EMIT STATUS function). However, with the /GO qualifier, the RUN/TASK function does not wait for the spawned task to terminate or emit status. Note you cannot access the symbols $RESULT and $STATUS for a task run with /GO.

/STATUS

This switch causes Flow to use the status emitted by the spawned task as the Flow status for the RUN operation. If the low bit of the status is clear (the status is even), Flow declares the error message:

```
TASK-E-TSKERR, Spawned task status is an error, STS
```

There is one parameter qualifier:

/CMD="command"

or

/COM="command"

This qualifier applies to the filespec parameter. /CMD allows you to directly pass information to the task you are invoking. Using /CMD requires that you understand the task's call interface. You can use CMD in a a menu (in FMS named data or FDT action strings) to allow a user-selected option to be passed to the executing task.

Steps in running a task:

1. Spawn as already installed.

2. If previous step fails and name is three characters long, spawn as ...xxx.

3. If previous step fails, install and spawn using the installed name. Remove when task completes.

P/OS Directive

PROTSK

Example

The following example invokes the EDT editor and passes EDT the name of the file to edit. Note that EDT is an installed task with the taskname ...EDT.

```
> RUN/TASK ...EDT/CMD="EDT myfile.dat"
```

3.53 SET DEFAULT

Change the default device and/or directory name.

Format

```
SET DEFAULT [ddnn:]dirspec
```

Where:

ddnn:	is the device name.
dirspec	is the directory name enclosed in square brackets.

Description

Flow applies the new default to all subsequent file specifications that do not explictly give a device or directory name.

P/OS System Routine

PROLOG

3.54 SET KEYPAD

Enable or disables function keys and keypad keys on the Professional 350 keyboard.

Format

SET [NO]KEYPAD

Description

If you specify SET KEYPAD, then Flow recognizes all function keys and keypad keys. If you specify SET NOKEYPAD, then Flow does not recognize these keys.

Note that SET [NO]KEYPAD has no effect while you are running the Dumb Terminal Emulator (DTE).

3.55 SET VERIFY

Control whether or not Flow displays command lines in command procedures.

Format

```
SET [NO]VERIFY [outputspec]
```

Where:

outputspec	is a file specification representing the file to which Flow sends the displayed command lines.

Description

By default, when Flow processes command procedures executed interactively, it does not display the command lines at the terminal screen. However, Flow always displays system responses and error messages.

SET VERIFY overrides the default setting, causing Flow to display all lines in command procedures during execution. Flow performs all symbol substitutions before displaying a line containing any symbols; thus you see only the equivalence strings for symbols.

When you change the verification setting, it remains in effect for all command procedures that you subsequently execute.

3.56 SHOW APPLICATION

Display the equivalence string associated with a defined application.

Format SHOW APPLICATION[/qualifier] appname

Where:

appname	is the name of a P/OS installed application, as you have defined it using the DEFINE/APPLICATION function.

Description

You define an application with the DEFINE/APPLICATION function.

The qualifier is:

/ALL

Shows all currently defined applications.

3.57 SHOW COMMAND

Display the equivalence string associated with a dynamic command. You define dynamic command with the DEFINE/COMMAND function.

Format

```
SHOW COMMAND[/qualifier] [dyncommand]
```

Where:

dyncommand	is the name of the dynamic command whose equivalence string you want to see.

Description

The qualifier is:

/ALL

Shows all currently defined dynamic commands.

Examples

```
> SHOW COM SPAWN
> SHOW COM/ALL
```

3.58 SHOW DEFAULT

Display the current default device and directory name.

Format

```
SHOW DEFAULT
```

Description

Flow supplies the current device and/or directory name whenever you omit them from a file specification.

P/OS System Routine

PROLOG

3.59 SHOW KEY

Display one or all key definitions.

Format

```
SHOW KEY[qualifier] [keynumber]
```

Where:

keynumber	is the keypad or function key value of the key that you have previously defined with DEFINE/KEY. Table 3-1 lists these values.

Description

If you omit the keynumber parameter, the SHOW KEY function displays all the currently defined keys.

The qualifier is:

/ALL

Shows all currently defined keys.

Examples

```
> SHOW KEY 19
> SHOW KEY/ALL
```

3.60 SHOW LOGICAL

Display the equivalence string associated with a logical name. You define a logical with the DEFINE/LOGICAL command.

Format SHOW LOGICAL logname

Where:

logname	is the name of the logical whose equivalence string you want to see.

Description

See DEFINE/LOGICAL for a description of the differences between logicals and symbols.

3.61 SHOW SYMBOL

Display the equivalence string associated with a symbol. You define a symbol with the DEFINE/SYMBOL command.

Format

```
SHOW SYMBOL[/qualifier] [symname]
```

Where:

symname	is the name of the symbol whose equivalence string you want to see. Do not specify symname if you specify the /ALL qualifier.

Description

The qualifier is:

/ALL

Specifies that Flow display all the current symbols in the symbol table.

3.62 SHOW TAG

Display one or all tags defined for Form Interface forms.

Format

```
SHOW TAG[/qualifier] [tagname]
```

Where:

tagname	is the tag name of an FI form.

Description

This function lists a tag you have defined with DEFINE/TAG. See Chapter 4 for details on tags.

The qualifier is:

/ALL

Displays all the currently defined tags.

3.63 TYPE OR SHOW FILE

Display the contents of a file or group of files on the terminal screen.

Format

```
TYPE filespec [,filespec]...
```

or

```
SHOW FILE filespec [,filespec]...
```

Where:

filespec	is the file specification for the file that you want to type.

Description

You can use the HOLD SCREEN key to control the output as it scrolls up the screen.

P/OS System Task

PIP.TSK

3.64 UNBLOCK

Unstop a task that was stopped with the BLOCK function.

Format

UNBLOCK instname

Where:

instname	is the installed name of the task that you are resuming.

Description

UNBLOCK is the counterpart of BLOCK; UNBLOCK/RESUME is the counterpart of BLOCK/SUSPEND.

P/OS System Directive

USTP$

Examples

```
> UNBLOCK SHAWN
```

3.65 UNBLOCK/RESUME

Resume a task that was suspended with the BLOCK/SUSPEND function.

Format

UNBLOCK/RESUME instname

Where:

instname	is the installed name of the task that you are resuming.

Description

UNBLOCK/RESUME is the counterpart of BLOCK/SUSPEND.

P/OS System Directive

RSUM$

Examples

```
> UNBLOCK/RESUME FLOW
```

3.66 UNWIND

Execute Flow's first function.

Format

```
UNWIND
```

Description

UNWIND is similar to EXIT in that both pop functions from the Flow stack. UNWIND, however, returns you all the way to the first function on the Flow stack (EXIT pops functions one at a time). You can set the first function by defining the symbol FLOW$_FIRST_FUNCTION. For example:

```
> DEFINE/SYMBOL FLOW$_FIRST_FUNCTION "MENU EM"
```

UNWIND operates by popping successive invocations off the Flow stack until reaching Flow's first function invocation. See Chapter 2 for a description the Flow stack.

3.67 WAIT

Suspend Flow for a number of seconds.

Format

```
WAIT secs
```

Where:

secs	is the number of seconds for which you want to suspend Flow.

Description

This function is useful in synchronizing separate Flow tasks.

Example

```
> WAIT 10
```

3.68 WRITE SYS$OUTPUT

Write a message to the terminal screen.

Format

WRITE SYS$OUTPUT message

Where:

message	is either a quoted literal string or a symbol (not in quotes or apostrophes).

Description

If you enclose the message in quotation marks, Flow prints the message exactly as it appears within the quotation marks. If you do not use quotation marks, then Flow attempts to translate the message as a symbol. You receive an error in this case if the message is not a defined symbol.

Examples

```
> WRITE SYS$OTUPUT  "This is a message string."
> WRITE SYS$OUTPUT  asymbol
```

CHAPTER 4

USING THE FORM INTERFACE FACILITY

The Form Interface Facility (FI) handles display and processing of all forms. PRO/Office Workstation allows you to use either of the following Professional Tool Kit utilities to develop your forms:

- Forms Management System (PRO/FMS)
- Frame Development Tool (FDT)

These utilities allow you to create, store, and modify forms.

All DIGITAL-supplied PRO/Office Workstation forms were designed using PRO/FMS. However, if you like you can provide FDT forms for your custom Workstation. Also, you can use generic forms, which are forms that you can partially customize using standard editors such as EDT and PROSE.

After describing the available types of forms, this chapter describes how to display forms, and how you use FMS, FDT, or a standard text editor to customize them. We have organized the latter part of the chapter according to what utility you decide to use in designing your forms:

- Section 4.3 describes how to customize FMS forms.
- Section 4.4 describes how to customize FDT forms.
- Section 4.5 describes how to customize generic forms.

(Note that the Tool Kit documentation contains complete descriptions of PRO/FMS and FDT. This chapter describes these utilities within the context of PRO/Office Workstation.)

The final section in this chapter describes how to write or customize help frames for PRO/Office Workstation.

4.1 TYPES OF FORMS

There are several levels of form classification in PRO/Office Workstation.

If you were to consider what a form's appearance is to the user, you could classify all PRO/Office Workstation forms into the following categories*:

- **Argument forms**

  This type of form typically has many active fields for input/output. It is suited to fetching or displaying symbol values, and for passing values to command procedures.

- **Menu forms**

  This form has one active field for input, usually called the CHOICE field. It is suited to displaying several options and allowing user to select one.

Were you to consider when you specify the description of the form's appearance, you could further classify forms into:

- **Static**

  You specify all of the form description when you create the form. Upon displaying the form, FI reads the description from a file stored on disk.

- **Dynamic**

  You specify most of the form description when you display the form. Upon displaying the form, FI reads the description from the command line you use to invoke FI. Some description is also stored on disk.

* The end-user documentation for PRO/Office Workstation calls an argument form a form, and calls a menu form a menu. In this manual, "form" has a broader meaning than in the User's Guide.

Finally, consider <u>how</u> you define and display the form:

- **A1/FMS**

  You define the form in the same way that VAX ALL-IN-1 does: you use FMS, and store a form-type directive in the named data. Also, you use the FMS form driver software to display the form. This type of form is always static.

- **FDT/FDT**

  You use FDT to define and display a form. This type of form is always static.

- **GENERIC/FMS**

  You use a standard text editor to define the form, and you use the FMS driver to display the form. This type of form is always static.

- **PARAMETER BLOCK/FMS**

  You define the form by packing information into a parameter block, and you display the form using the FMS driver. This type of form is always dynamic.

Figure 4-1 illustrates the classification of forms, showing the categories provided by PRO/Office Workstation. Sections following the figure describe the two major categories; the remainder of the chapter describes the sub-categories.

TYPES OF FORMS

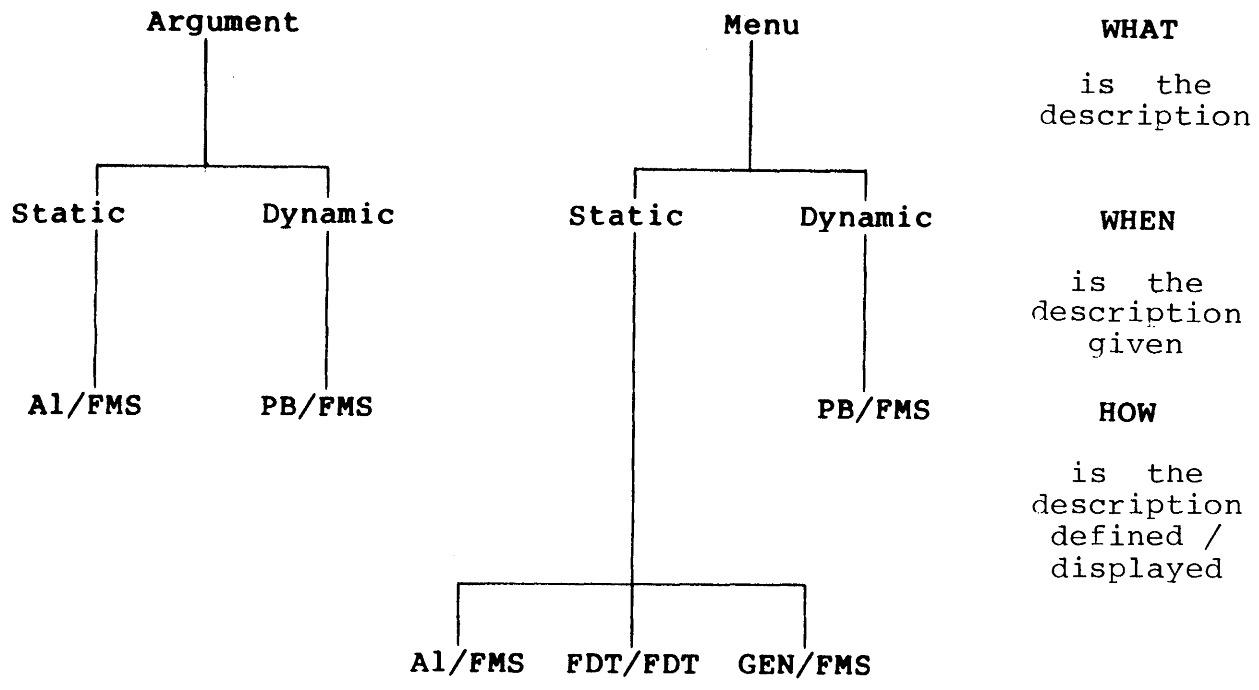
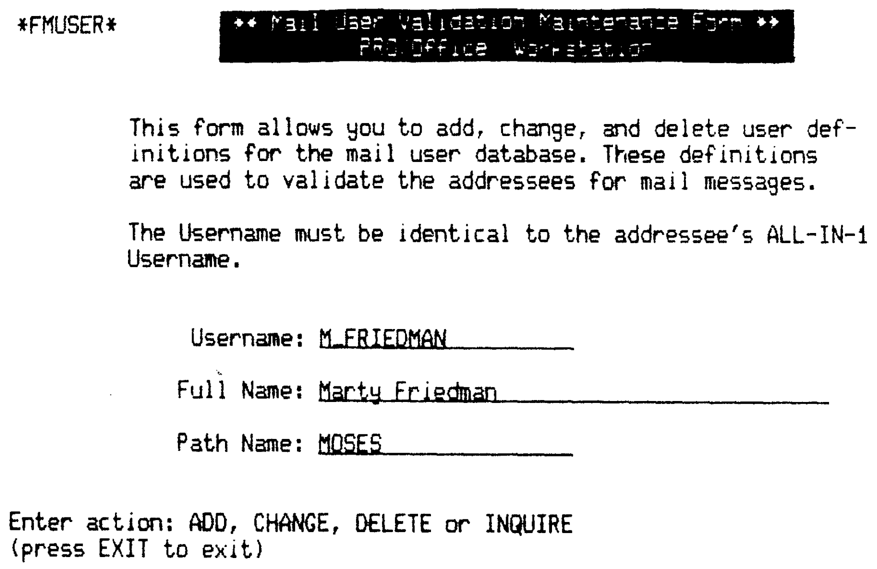


Figure 4-1: Classification of Forms

4.1.1 Argument Forms

An argument form has many active fields for input and output. It is suited to fetching or displaying values to assign to a symbol, and for passing values to command procedures. From the user's point of view, an argument form is meant to "be filled out," in the way that one might fill out a loan application or questionnaire. Of course, the difference here is that this is an electronic form.

Figure 4-2 illustrates a typical argument form. The figure shows a form that accepts setup information from a user. Note that PRO/Office Workstation uses argument forms for all of its setup modes.



```
*FMUSER*          ** Mail User Validation Maintenance Form **
                           PRO/Office  Workstation

          This form allows you to add, change, and delete user def-
          initions for the mail user database. These definitions
          are used to validate the addressees for mail messages.

          The Username must be identical to the addressee's ALL-IN-1
          Username.

               Username: M_FRIEDMAN
              Full Name: Marty Friedman
              Path Name: MOSES

Enter action: ADD, CHANGE, DELETE or INQUIRE
(press EXIT to exit)
```

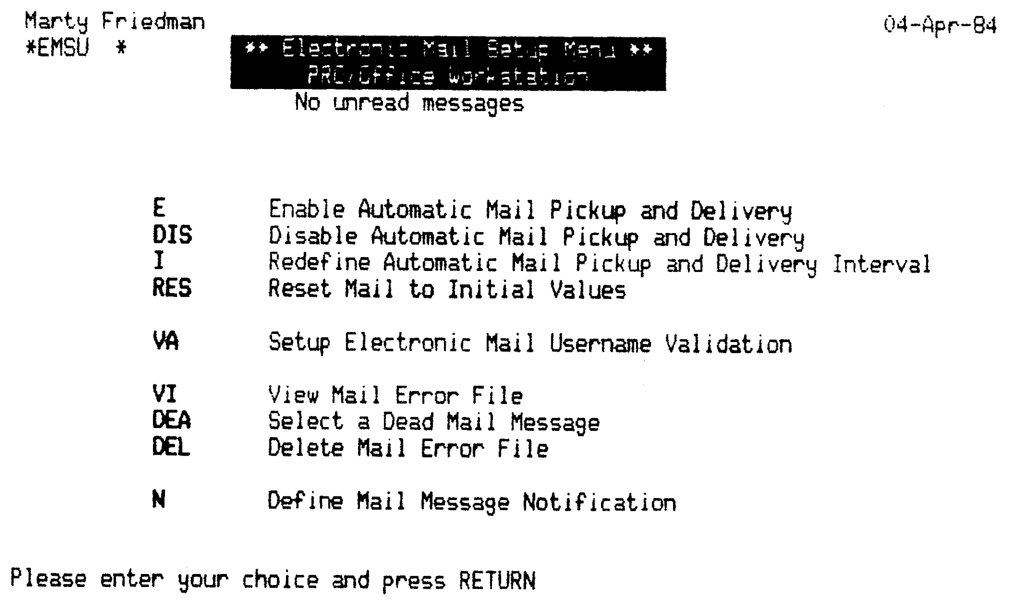
Figure 4-2: Example of an Argument Form

You display an argument form using the FORM or FIELD functions in Flow.

4.1.2 Menu Forms

Menu forms have one active field for input. The form displays several options, one of which the user chooses. The active field, usually called the CHOICE field, accepts the user's choice. Menu forms are most useful in defining menu trees, which guide the user through a series of options.

Figure 4-3 illustrates a typical menu form.



```
Marty Friedman                                                        04-Apr-84
*EMSU  *         ** Electronic Mail Setup Menu **
                    PRC/Office Workstation
                     No unread messages


          E        Enable Automatic Mail Pickup and Delivery
          DIS      Disable Automatic Mail Pickup and Delivery
          I        Redefine Automatic Mail Pickup and Delivery Interval
          RES      Reset Mail to Initial Values

          VA       Setup Electronic Mail Username Validation

          VI       View Mail Error File
          DEA      Select a Dead Mail Message
          DEL      Delete Mail Error File

          N        Define Mail Message Notification

Please enter your choice and press RETURN
```

Figure 4-3: Example of a Menu Form

4.2 DISPLAYING FORMS

You request FI to display a form by issuing the MENU, FORM, or FIELD function. Use the FORM or FIELD functions only to display an argument form. Also, use the MENU function to display only a menu form. The format of these functions is:

```
func[/qualifier...] formspec [fieldname[/qualifier]]
```

Where:

func — is MENU, FORM, or FIELD.

formspec — is a form specification, which indicates the desired form in the format:

```
[tag[$[formname]]]
```

You can specify just the tag, just the formname, or both delimited by a dollar sign. If you specify just the formname (that is, you omit tag$), FI supplies the default tag, called DEFAULT. Section 4.2.1 describes tags.

You specify field name only for the FIELD command. See Chapter 3 for further details on these functions and their qualifiers.

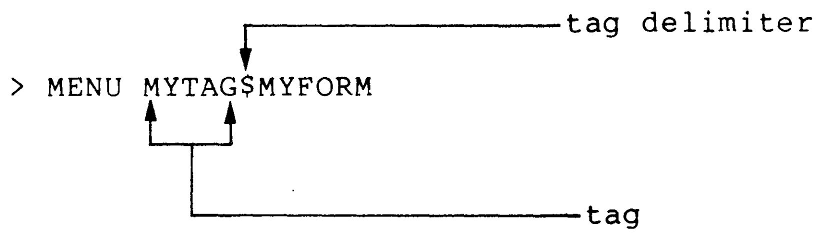
Examples of the functions follow. Note that MAIN and XYZ in the first two examples could be either formnames or tags. In the second two examples, both MAIN and XYZ must be formnames.

```
> MENU MAIN
> FORM XYZ
> MENU MYTAG$MAIN
> FORM MYTAG$XYZ
> FIELD MYTAG$ABC
```

4.2.1 Specifying Tags

The Form Interface Facility uses a tag as part of any form specification. A tag is a symbol whose equivalence contains information that FI uses to perform a request. FI reads the tag from the form specification when you invoke the MENU, FORM, or FIELD command.

The following diagram shows the position of a tag and a tag delimiter within a form specification.



```
> MENU MYTAG$MYFORM
```

The tag supplies such information as:

- How you have defined the form (/DEFINITION=defval).
- How you will display the form (/DISPLAY=disval).
- The name of the form library containing the form (/LIBRARY=libname).

You specify this information when you define a tag, using the DEFINE/TAG Flow function. The format of DEFINE/TAG is:

```
DEFINE/TAG tagname equivalence
```

Where:

tagname	is the name of the tag you are defining. Do not include the dollar sign ($) in a tag; this symbol delimits a tag from a formname in a complete form specification.
equivalence	is a series of qualifiers that provide information to FI.

Table 4-1 lists all the tag qualifiers that you can specify in the equivalence. Note that you can abbreviate the qualifiers to their minimally-unique names.

Table 4-1: Tag Qualifiers

Tag Qualifier	Description
/DEFINITION=defval	Specifies how you define the form (defval): ● AI -- You define the form using the FMS utilities FED and FUT. The named data of the form contains either a .MENU or .ARG form-type directive. The form description is stored statically in named data. ● FDT -- You define the form using FDT. The form description is stored statically in the frame description file.

Tag Qualifier	Description
(Continued.)	● GEN -- You define the form using a standard text editor. The form description is stored statically in an ASCII file. ● PB -- You define the form in a parameter block. FI reads the form description from qualifiers you specify in the CLI command line. You use the CALL function to display the form.
/DISPLAY=disval	Specifies how you display the form (disval): ● FMS -- You display the form using the FMS form driver. ● FDT -- You display the form using FDT.
/LIBRARY=libname	Specifies the name of the library (libname) containing the specified form. You must include the filename's extension (for example, .FLB for an FMS form).
/FORM=formname	Specifies the name of the form that you want to display. The formname must <u>not</u> include a file extension (such as .FRM).
/FILE=filename	For generic forms only. Specifies the name of the generic description file that describes your form.

Examples of tag definitions follow.

```
> DEFINE/TAG  INPUT     "/DEF=Al/FORM=INPUT"
> DEFINE/TAG  DISPLAY   "/DEF=Al"
> DEFINE/TAG  HELP      "/DIS=FDT/LIB=MYHELP.HLP"
> DEFINE/TAG  TEST      "/FORM=TEST/LIB=MYLIB.FLB"
```

If you do not specify a tag in a form specification, FI supplies a default tag, DEFAULT. Consequently, FI interprets the Flow command:

```
> MENU MYFORM
```

as

```
> MENU DEFAULT$MYFORM
```

PRO/Office Workstation defines the tag DEFAULT as follows:

```
> DEFINE/TAG  DEFAULT
"/DEF=A1/DIS=FMS/LIB=DW1:[ZZFLOW]OAFMS.FLB"
```

4.2.2 Precedence of FI Qualifiers

FI qualifiers are those qualifiers that affect the action of the Form Interface. These include qualifiers you can specify on tags, on an FI request, or in the named data/action strings of a static form. An FI request is simply the Flow command that you use to display a form.

In some cases, you can specify the same FI qualifier in two different places. For example, suppose you define a tag as follows:

```
> DEF/TAG SAMPLE  "/DEF=A1"
```

Then, suppose you invoke a form with the command:

```
> FORM/DEF=FDT SAMPLE$MYFORM
```

In this example, you have specified the /DEFINITION qualifier in two places: the tag and the request. Here the definition you gave in the request overrides the definition you gave in the tag.

Here's another example. Suppose you have created an FMS static form whose named data contains the following information:

Name	Data
.MENU	/ALLOW/EXACT
WP	MENU WP
DM	MENU DM
EM	MENU EM

Then, suppose you displayed the form using the following Flow function:

```
> MENU/NOALLOW/NOEXACT myform
```

You have specified the FI qualifiers /[NO]ALLOW and /[NO]EXACT in both the static area (named data) of the form, and in the request. Again, the qualifier in the request prevails.

The following list shows the precedence of FI qualifiers, in order of declining precedence from top to bottom:

1. Qualifier in the request.

2. Qualifier in the tag.

3. Qualifier in the static area (named data for FMS forms).

By providing this precedence, FI allows you to override a particular qualifier at various levels when displaying a form.

4.2.3 Displaying Forms Statically (/DEF=A1, FDT, or GEN)

Displaying a form statically means that the form description used by the Form Interface Facility is stored on disk:

- For FMS ALL-IN-1 type forms (/DEF=A1), the form description is stored in a form library. To display an FMS form statically, the form description must contain a named data section that completely describes all options on the form. Section 4.3 describes FMS forms.

- For FDT forms (/DEF=FDT), the form description is stored in a frame definition file. To display an FDT form, the frame definition must contain action strings that completely describe all options on the form. Section 4.4 describes FDT forms.

- For generic forms (/DEF=GEN), the form description is stored partly in an FMS or FDT library and partly in a standard ASCII file, called a generic description file. This file must contain the TEXT and OPTION directives completely describing all options on the form. Section 4.5 describes generic forms.

To display a form statically, simply invoke one of the commands MENU, FORM, or FIELD, specifying the name of a form. As long as the form's description is stored in an appropriate disk file, FI displays the form statically.

4.2.4 Displaying Forms Dynamically (/DEF=PB)

To display a form dynamically, you must "manually" pack the parameter block that Flow uses to communicate with FI. A parameter block is simply a piece of memory that software modules use to communicate with each other.

You pack the parameter block by passing information directly to FI from the Flow command line. To do this, you use the CALL function instead of the MENU, FORM, or FIELD functions.

The information you supply in the parameter block includes:

- Request (MENU, FORM, FIELD)
- Definition is PB (/$DEF=PB)
- Library name
- Form name
- Contents of each field

The format of the CALL command when used to display a form dynamically is:

```
CALL SWB$FI/qualifier...
```

Where:

SWB$FI	is the installed name of the Form Interface task.
/qualifier	must include the qualifiers listed below.

The qualifiers you can specify for either dynamic argument forms or dynamic menu forms follow. Sections 4.2.4.1 and 4.2.4.2 describe the qualifiers that are unique to each type of form.

/$REQUEST=reqname

This qualifier specifies the request that you want FI to process. If you are displaying an argument form, use /$REQ=FORM or /$REQ=FIELD. If you are displaying a menu form, use /$REQ=MENU.

/$DEFINITION=PB

This qualifier has the same purpose as the equivalent tag qualifier described in Table 4-1. To display a form dynamically, you must specify /$DEFINITION=PB

/$DISPLAY=disval

This qualifier has the same purpose as the equivalent tag qualifier described in Table 4-1. The value of disval can only be FMS.

/$LIBRARY=libname

This qualifier has the same purpose as the equivalent tag qualifier described in Table 4-1. PRO/Office Workstation supplies an FMS form for dynamic menu forms, called FMLIST. This form is in library OAFMS.FLB. For dynamic argument forms, you must create your own form description.

/$FORM=formname

This qualifier has the same purpose as the equivalent tag qualifier described Table 4-1. PRO/Office Workstation supplies an FMS form for dynamic menu forms, called FMLIST. This form is in library OAFMS.FLB. For dynamic argument forms, you must create your own form description.

4.2.4.1 Dynamic Argument Forms - For a dynamic argument form, you must create a template form using FMS. (FDT argument forms are not available.) However, you do not describe the form's fields in the template form, as you would in a static argument form. Instead, use the following qualifiers:

/PUT=fldnme,symnme

Where:

fldnme	is the name of a field on the form used for output.
symnme	is the name of a symbol whose equivalence value you want to display.

This qualifier allows you to display the equivalence value of the symbol symnme in the field fldnme.

/GET=fldnme,symnme

Where:

fldnme	is the name of a field on the form used for input.
symnme	is the name of a symbol to which the user will assign an equivalence value.

This qualifier allows the user to enter an equivalence value in the field fldnme, which FI assigns to the symbol symnme.

/OUTPUT=fldnme,string

Where:

fldnme	is the name of a field on the form used for output.
string	is a character string surrounded by quotation marks.

This qualifier allows you to display the specified string in the field fldnme.

/INPUT=fldnme

Where:

fldnme	is the name of a field on the form used for input.

This qualifier allows you to read the user's input as a character string from the field fldnme.

An example of dynamic argument forms invocation follows. Note that you cannot specify /$DIS=FDT.

```
> CALL SWB$FI/$REQ=FORM/$DIS=FMS/$DEF=PB/$LIB=OAFMS.FLB -
_/$FORM=PROFIL/$INPUT=DEPT
```

4.2.4.2 Dynamic Menu Forms - PRO/Office Workstation provides a template menu form in OAFMS.FLB, called FMLIST. (FDT dynamic forms are not available.) You supply the menu options in command line qualifiers, and FI writes the options on the form.

To specfify the options on the command line, use the following format:

/#option

Where:

option	is a string that FI dynamically places on the menu as an option.

FI places the first option you specify in option 1, the second option in option 2, and so on.

An example of the CLI command line follows. Note that you cannot specify /$DIS=FDT. Also, note that the /PREFIX qualifier specifies a prefix string for parameters returned by FI. Thus, you could find out the user's choice by displaying the symbol fmlist$CHOICE.

```
> CALL SWB$FI/$REQ=MENU/$DIS=FMS/$DEF=PB/$LIB=OAFMS.FLB -
_/$FORM=FMLIST/#DTE/#DIR/#EXIT/PREFIX=fmlist
```

4.3 CUSTOMIZING FMS FORMS

An FMS form consists of static text and variable fields. You use the FMS Form Editor (FED) to create a form, and to insert the text and fields. FED also lets you assign form-wide display attributes, as well as particular attributes for each field. Additionally, you specify named data in a form. Named data allows you to invoke Flow functions from an FMS form. (Section 4.3.3 describes named data.)

PRO/Office Workstation stores all forms in a form library. A form library is a file that contains form description files and a directory of names for each form description. PRO/Office Workstation supplies you with the following form libraries:

OAFMS.FLB	contains all forms that serve as the PRO/Office Workstation end-user interface. This is an FMS library.
OAFDT.FLB	contains FDT forms that you might want to use. You can place your own FDT forms here.

This library resides in the directory into which you installed PRO/Office Workstation. Execute the CLI command SHOW SYMBOL FLOW$_EXE_ACCOUNT to find the name of this directory.

The Tool Kit contains the Forms Editor (FED) and the Forms Utility (FUT), which you use to customize FMS forms and manipulate FMS libraries.

4.3.1 Executing FED and FUT

You can execute FED and FUT directly from Flow CLI, provided you have installed the PRO/Tool Kit. The following steps show how to use CLI to define a dynamic command that executes FED (follow the same steps for FUT):

1. Determine what directory contains the PROFED task by entering the CLI command:

```
> DIRECTORY [*]PROFED.TSK
```

2. Suppose you found that PROFED.TSK is installed in directory [ZZAP00008]. Use this to define a dynamic command that works from any directory:

```
> DEFINE/COMMAND FED "RUN/TASK [ZZAP00008]FED.TSK"
```

3. To execute FED from Flow CLI, simply type:

   ```
   > FED
   ```

Note that executing FED requires that your terminal "look like" a VT100. If you get a FED error message saying that you must use a VT100, execute the following Flow function:

```
> RUN/TASK [ZZFLOW]SETVT
```

See the Developer's Tool Kit document set for complete details on FMS.

4.3.2 Creating an FMS Form: A Sample Session

The following sample session describes how you use the form template MENU.FRM (supplied in the library OAFMS.FLB) as the basis of a new form. Using the template is probably the easiest and quickest means of creating a custom FMS form. If you are not very familiar with FMS, this sample session should help you get started.

1. Set your default directory to [ZZFLOW]:

   ```
   > SET DEF [ZZFLOW]
   ```

2. Run PROFED. (You can do this from CLI by executing the dynamic command FED as described in Section 4.3.1.

3. When you receive the FED> prompt, enter the name of the default FMS form library, OAFMS.FLB, and press RETURN.

4. FED prompts you for a formname; enter MENU and press RETURN.

5. When you see the COMMAND: prompt, enter the FORM command to allow you to define form-wide attributes, then press RETURN.

6. Now you can change the name of the form from MENU to whatever name you want. (Do not worry about losing MENU.FRM; this procedure preserves the original template for later use.) For this example use the name TEST. Simply type TEST over the name MENU, then press the TAB key to advance to subsequent fields in the display. Change any attributes that you want for your new form.

7. Return to the COMMAND: prompt by pressing the RETURN key. Now you can edit the new form. Enter EDIT after the COMMAND: prompt, then press RETURN.

8. Add your menu items to the new form while in FED edit mode.

9. When you have entered your menu items, return to the COMMAND: prompt by pressing the GOLD key and then the 7 key on the keypad.

10. Assign field attributes, or just look at the form's field names, by entering ASSIGN ALL at the COMMAND: prompt and pressing RETURN. FED displays the field name and prompts you for attributes. For each field, you can change field attributes by pressing the TAB key to advance to subsequent attributes. You can process the next field by pressing the RETURN key. When you have processed the last field, FED returns you to the COMMAND: prompt.

11. Place the named data for each menu option in the form by specifying NAME when you have the COMMAND: prompt, then pressing RETURN. For each option:

    - Enter the option name (under the name column) and press the TAB key once to go to the next field.

    - Enter the Flow function (under the DATA column) that you want to perform when the user selects the associated option.

    The .MENU directive contained in named data for the MENU.FRM form specifies default qualifiers. Change these only if you do not want the specified qualifiers. For further information on named data, see Section 4.3.3.

12. Now enter the SAVE command to save the new form as TEST.FRM. Press RETURN. Exit FED by entering CTRL-Z when you get the FED> prompt.

13. You must still enter the form in a library. Invoke FUT by executing a dynamic command as described earlier in this chapter.

14. When you receive the FUT> prompt, enter the command:

    ```
    FUT> OAFMS.FLB=OAFMS.FLB,TEST.FRM/RP
    ```

This loads the form TEST.FRM into the form library OAFMS.FLB. Exit FUT by entering CTRL-Z when you get the FUT> prompt.

15. Now you can call up your new menu, TEST, by invoking the MENU function from CLI:

```
> MENU TEST
```

For this invocation, FI uses the default tag, DEFAULT.

4.3.3 Specifying FMS Named Data (/DEF=Al, /DIS=FMS)

Named data is a means of entering non-FMS data within an FMS form. Named data consists of information that is associated and stored with a form, but not displayed in the form.

FMS only stores and manages named data, passing it to FI for interpretation. For example, suppose you create a simple menu with three choices, as illustrated in Figure 4-4. The figure shows the menu as you would see it while in FED's EDIT mode, and the named data as you would see it while in FED's NAME mode.

When you display the form shown in Figure 4-4, FMS passes the named data to FI. In turn, FI interprets the named data. The .MENU directive in the name section tells FI what type of form it displays. The data associated with .MENU tell FI what fields are active, and what the contents of each field should be. Below the lines containing the .MENU information, the named data describes the menu options and what Flow function to perform for each option.

Whenever you use named data in an FMS form, you are defining a static, ALL-IN-1 type form (/DEF=Al). To distinguish between argument forms and menu forms, you use either the .ARG directive or the .MENU directive in the named data of a form. Sections following the figure describe each type of form and the associated directive.

```
XXXXXXXXXXXXXXXXXXXXXX                                      XXXXXXXXX
PRO/Office Workstation                                         XXXXXX

                          d   i   g   i   t   a   l

                                ** Sample Menu **
                              PRO/Office Workstation
                              XXXXXXXXXXXXXXXXXXXXXX


                      WP      Document Processing
                      EM      Electronic Mail
                      DM      Desk Management


                      Please enter selection and press RETURN:
                      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
COMMAND:
                              Named Data Entry Form


 Name                           Data
|----|  |-----------------------------------------------------------|
.MENU   /ALL/GET=USER,OA$USER/GET=DATE,OA$DATE/GET=TIME,OA$TIME
.MENU1  /GET=MAIL,OA$MAIL_COUNT_DISPLAY
WP      MENU WP
DM      MENU DM
EM      MENU EM
```

Figure 4-4: Simple Three-Choice Menu and Named Data

4.3.4 FMS Static Argument Forms

You specify the .ARG directive in named data to indicate a static argument form.

Format (Named Data)

```
 Name                            Data
|----|  |-----------------------------------------------------|
.ARG    /[NO]KEYPAD
fldnme  [/qualifier...]
.
.
.
```

Where:

fldnme	is the name of a field on the form. You can specify as many field names as there are fields on the form.

Description

Use the static argument form primarily to perform field processing, passing data through fields either to or from a symbol. Note that the INQUIRE function in Flow performs a similar operation.

The qualifiers are:

/[NO]KEYPAD

/KEYPAD enables FI to interpret the escape sequences generated by the numeric keypad. /NOKEYPAD disables FI from interpreting these escape sequences. You can specify this qualifier only in the data section of the .ARG directive.

/UPPER

Converts to uppercase all data the user enters in the associated field.

/GETSAVE=symbol

Place a value from the specified symbol into the associated field.

/PUTSAVE=symbol

Place a value from the associated field into the specified symbol.

/BLANK

Inhibits /GETSAVE on a non-blank field. That is, if a value currently resides in a field, this qualifier prohibits the user to enter another value into that field. This qualifier is useful when you have defined a default value for the field, using a FED field attribute.

/CLEAR

Erase the value in the associated field before processing a new value. Use this qualifier with /PUTSAVE. It clears the field before allowing the user to enter any values into that field.

Example

```
 Name                        Data
|----|  |-------------------------------------------------|
.ARG    /KEYPAD                 ;function key sequences enabled
fld1    /GET=sym1               ;place value of sym1 into fld1
fld2    /BLANK                  ;inhibit /GET if field is nonblank
fld3    /PUT=sym3               ;place value of fld3 into sym3
fld4    /CLEAR                  ;clear the field for each display
fld5    /PUT=sym5/UPPER         ;uppercase input before assignment
fld6    /BLANK/GET=sym6         ;don't allow default to be changed
```

4.3.5 FMS Static Menu Forms

You specify the .MENU directive in named data to indicate a static menu form.

Format (Named Data)

```
 Name                            Data
|----|  |---------------------------------------------------|
.MENU   /qualifier...
[.MENUn /qualifier...]
.
.
.
optnme  function
.
.
.
```

Where:

optnme	is the name of a menu option on the form. You can specify as many option names as there are options on the form. function is the Flow function to be associated with the corresponding menu option. You can invoke a command procedure here to effectively invoke more than one function.
.MENUn	is a continuation of the .MENU directive, with the n incremented by 1 for every additional line. In the first .MENUn, n=1.

Description

You can use the static menu form as part of a menu tree, allowing the user to select functions.

The qualifiers are:

/[NO]KEYPAD

/KEYPAD enables FI to interpret the escape sequences generated by the numeric keypad. /NOKEYPAD disables FI from interpreting these escape sequences. You can specify this qualifier only in the data section of the .ARG directive.

/[NO]EXACT

/EXACT means that the user must enter the menu option exactly as it appears on the menu. /NOEXACT means that the user can enter a substring of the menu option, or can enter the full option with trailing characters (FI ignores the trailing characters).

/[NO]ALLOW

ALLOW means that if the user selects a menu option that doesn't exist on your menu, FI passes the user's input to Flow control for processing. NOALLOW means that FI intercepts nonexistent menu options and returns an error message.

/GET=fldnme,symnme

This qualifier causes FI to display the equivalence value of the symbol symnme in the field fldnme. This is useful in displaying values in the form that can change, such as OA$USER and OA$TIME.

/CHOICE=fldnme

This qualifier specifies the name of the field (fldnme) from which FI will read the user's choice.

Example

```
 Name                           Data
|----|  |----------------------------------------------------|
.MENU   /KEYPAD/GET=USER,OA$USER/GET=DATE,OA$DATE/CHOICE=CHOICE
.MENU1  /GET=TIME,OA$TIME/GET=MAIL,OA$MAIL_COUNT_DISPLAY
WP      MENU OAFMS$WP
DM      MENU OAFMS$DM
EM      MENU OAFMS$EM
CMD     COMMAND SYS$COMMAND
END     END
```

4.4 CUSTOMIZING FDT FORMS

The Tool Kit contains the Frame Development Tool, which you use to customize FDT forms. You can use FDT to create and display only static menu forms; it supports neither dynamic forms nor argument forms. You can also use FDT to create help frames.

4.4.1 Executing FDT

You can execute FED and FUT directly from Flow CLI, provided you have installed the PRO/Tool Kit. The following steps show how to use CLI to define a dynamic command that executes FED:

1. Determine what directory contains the FDT task by entering the CLI command:

   ```
   > DIRECTORY [*]FDT.TSK
   ```

2. Suppose you found that FDT.TSK is installed in directory [ZZAP00008]. Use this to define a dynamic command that works from any directory:

   ```
   > DEFINE/COMMAND FDT "RUN/TASK [ZZAP00008]FDT.TSK"
   ```

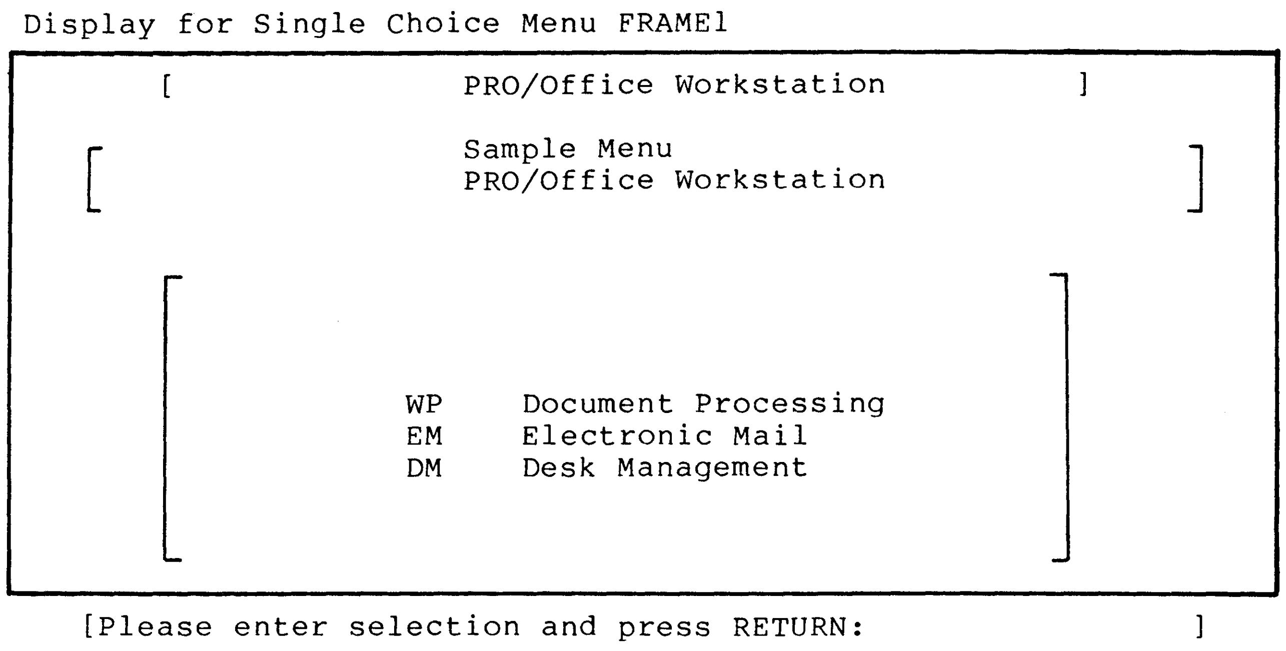
3. To execute FDT from Flow CLI, simply type:

   ```
   > FDT
   ```

4.4.2 Specifying FDT Action Strings

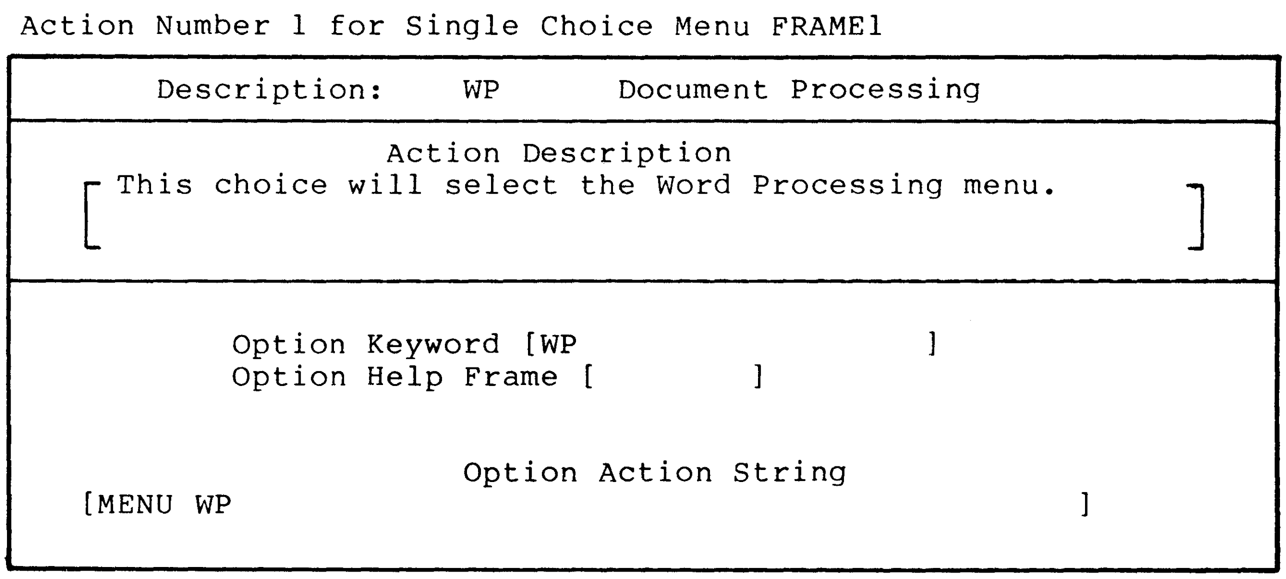
Action strings are part of menu's frame definition, and are a means of entering non-FDT data within an FDT form. An action string consists of information that is associated and stored with a form, but not displayed in the form.

FDT only stores and manages actions strings, passing them to FI for interpretation. For example, suppose you create a simple menu with three choices. Figure 4-5 shows such a menu as you would see it in on the DISPLAY form. Figure 4-6 shows an action string for the first choice as you would see it on the ACTION form.



```
Display for Single Choice Menu FRAME1

     [                PRO/Office Workstation                ]

                      Sample Menu
  [                   PRO/Office Workstation                     ]



     [                                                     ]
                   WP    Document Processing
                   EM    Electronic Mail
                   DM    Desk Management


  [Please enter selection and press RETURN:                      ]
```

Figure 4-5: **FDT DISPLAY Form for Three-Choice Menu**



```
Action Number 1 for Single Choice Menu FRAME1

      Description:    WP      Document Processing

                    Action Description
  [ This choice will select the Word Processing menu.            ]


          Option Keyword [WP                 ]
          Option Help Frame [       ]

                      Option Action String
  [MENU WP                                                  ]
```

Figure 4-6: **FDT ACTION Form**

4.5 CUSTOMIZING GENERIC FORMS (/DEF=GEN)

A generic form is a form that consists partly of an FMS form and partly of an ASCII description file . The FMS part of the form is static; you cannot change it unless you use the appropriate FMS editor. The description file, however, is a text file that you can edit with any standard word processor, such as EDT or PROSE.

The description file contains Form Interface directives and menu text. (You can create only menu forms generically, not argument forms.) The FI directives tell FI what options appear on the menu and what actions to take when the user makes a selection. The Form Interface functions that you can place in the description file are:

- **[NO]ALLOW**

  ALLOW means that if the user selects a menu option that doesn't exist on your menu, FI passes the user's input to Flow control for processing. NOALLOW means that FI intercepts nonexistent menu options and returns an error message.

- **[NO]EXACT**

  EXACT means that the user must enter the menu option exactly as it appears on the menu. NOEXACT means that the user can enter a substring of the menu option, or can enter the full option with trailing characters (FI ignores the trailing characters).

- **TEXT**

  TEXT has one required parameter:

  TEXT "mnuopt"

  Where:

mnuopt	is the name of the menu option as it appears on the form.

- **OPTION**

  OPTION has two required parameters:

  OPTION   mnuopt   flowfun

  Where:

mnuopt	is the name of the menu option as FMS receives it. This is the name part of named data in an FMS form.
flowfun	is the actual Flow function to perform when the user selects the corresponding mnuopt. This is the data part of named data in an FMS form. in FDT.

- **FORM**

  FORM has two required parameters:

  FORM formnme   libnme

  Where:

form	is the name of the FMS form that that FI uses as the template.
libnme	is the name of the FMS library containing the template form.

You can also use an exclamation mark (!) to delimit comments.

Figure 4-7 shows an example generic description file.

```
!---------------------------------------------------------------
!       This is a sample description file for a generic form. The
!       following line specifies the form and library that FI uses.
!       This function is optional because you could use a tag name
!       in the MENU command instead.
FORM  GEN.DAT    OAFMS.FLB

!       ALLOW means that if you enter a menu option that doesn't
!       exist, FI passes your entry to Flow for processing.
!       NOEXACT means that if you can type a substring or
!       "superstring" of the actual menu option.
ALLOW
NOEXACT

!       FI displays the quoted data as menu options on the form.
!       You can use up to 12 fields. FI ignores excess text lines.
TEXT "ED        Run the EDT editor"
TEXT "PIP       RUN the Peripheral Interchange Processor, PIP"
TEXT "INS       Install another Flow task with the name F"
TEXT "FLOW      Run the Flow task named F"
TEXT "EXIT      Invoke the EXIT function"
TEXT "A1        Display the menu in BBB$MENU"

!       The ENDTEXT specification tells the program that there are
!       no more text lines after this point. This line is optional;
!       however, it does help performance if you use it.
ENDTEXT

!       The option lines translate the user's selection to a Flow
!       function. There is no limit to the number of option lines.
OPTION  ED      "EDT"
OPTION  PIP     "RUN PIP"
OPTION  INS     "INS FLOW/NAME=F"
OPTION  FLOW    "RUN F"
OPTION  EXIT    "EXIT"
OPTION  A1      "MENU BBB$MENU"
!---------------------------------------------------------------
```

Figure 4-7: Example Generic Description File

4.5.1 Displaying Generic Forms

To display a generic form, you use the MENU Flow function. However, instead of specifying a formname, you specify the name of the generic description file.

Also, you need to specify particular values for serveral of the tag qualifiers. For example, you must use a tag that tells FI that you want to display a generic form; otherwise, FI would supply the default tag, DEFAULT, which indicates that you want to display only a standard FMS form.

You can define a tag for displaying a generic form as follows:

```
> DEFINE/TAG GENERIC -
"/DEF=GEN/DIS=FMS/LIB=[ZZFLOW]OAFMS.FLB/FOR=GEN"
```

Then you can display a form whose description file is MYFILE.TXT by invoking the command:

```
> MENU GENERIC$MYFILE.TXT
```

The tag GENERIC tells FI that you are displaying a generic form, that FMS actually displays the form, and where to look for the form.

4.6 CUSTOMIZING HELP FRAMES

The kit that you receive contains help frames in the following FDT library:

[ZZFLOW]OAHELP.FLB

This library contains one help frame for each form that is in [ZZFLOW]OAFMS.FLB. Each help frame has the same name as the form from which it is called. (See Appendix B for a list of form names supplied with the kit.)

Also, the library contains a general frame describing the syntax of all the CLI functions. This frame is called CLI.

FI automatically opens [ZZFLOW]OAHELP.FLB as the <u>default help library</u>; consequently, you must place all of your <u>custom help frames</u> in this library. When the user requests help, FI always searches [ZZFLOW]OAHELP.FLB to find the appropriate frame.

CUSTOMIZING HELP FRAMES

To call a help frame from an FMS form, you must use the directive

.HELP.

in the name section of named data. The corresponding data section for the .HELP. directive must contain the name of the help frame in [ZZFLOW]OAHELP.FLB that FI will display.

The following example illustrates the named data entry form for a menu called MAIN. The first line of named data contains the .HELP. directive and its corresponding help frame, MAIN.

Note that you should generally give the same names to a form and its corresponding help frame.

Named Data Entry Form

```
 Name                              Data
|----|  |-------------------------------------------------------------|
.HELP.  MAIN
.MENU   /ALL/GET=USER,OA$USER/GET=DATE,OA$DATE/GET=TIME,OA$TIME
.MENU1  /GET=MAIL,OA$MAIL_COUNT_DISPLAY
WP      MENU WP
DM      MENU DM
EM      MENU EM
```

For information on writing the help frames and storing them in a library, see the FDT documentation in the Tool Kit document set.

CHAPTER 5

USING THE DOCUMENT SERVICES FACILITY

The Document Services Facility is a database management system that manages user documents. The facility has its own functions that you can invoke by using the DOC Flow function. This chapter describes the organization of the database, as well as the functions provided by the Document Services Facility.

5.1 ORGANIZATION OF THE DOCUMENT SERVICES DATABASE

Document Services provides a default database that consists of the following files:

- **DW1:[ZZDOC0]DEFAULT.DDB**

  This is the RMS index for the database, which users know as the <u>default file cabinet</u>. The user can create alternative file cabinets if desired. A file cabinet is an ISAM file that contains one or more records representing headers for the user's documents. Document Services locates a document by searching a file cabinet for the document header.

- **DW1:[ZZDOC0]nnnnnnnnn.OAD**

  This is the P/OS file name for each actual document, where nnnnnnnnn is a zero-filled, decimal integer representing the document's order of creation. For example, a user's document files have the names 000000001.OAD, 000000002.OAD, 000000003.OAD, and so on.

Figure 5-1 illustrates the relationship between the DEFAULT.DDB file and the .OAD files. The figure shows a database containing three data files (000000001.OAD, etc.) and an index (DEFAULT.DDB) containing three headers for those data files.

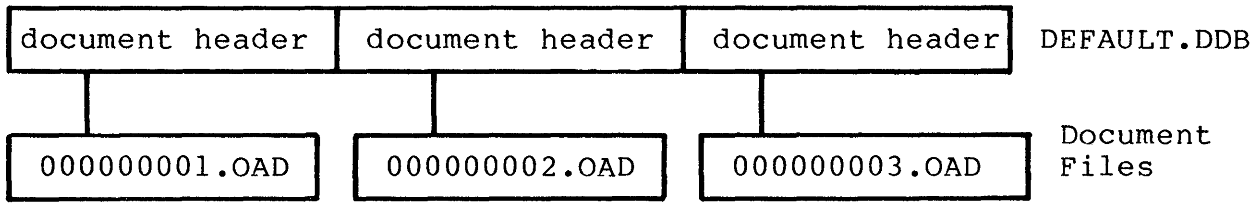


Figure 5-1: Document Services Database Organization

Each document header (record) in DEFAULT.DDB contains keys that enable Document Services to locate the desired document. The primary key for any document is its reference number, a number that Document Services assigns during document creation. Secondary keys are the folder name and the document title.

The following record definition illustrates the structure of a document header stored in DEFAULT.DDB. The maximum size of the header is 416 bytes (216 fixed bytes plus 200 variable bytes for the last field).

```
05      dsh$_refnum              (04) longword (key 0)
05      dsh$_folder              (16) ascii (key 1)
05      dsh$_title               (60) ascii
        10      dsh$_title_key   (20) ascii (key 2)
        10      dsh$_title_ext   (40) ascii
05      dsh$_pointer             (04) longword (key 3)
05      dsh$_filnam              (40) ascii
05      dsh$_keyword             (20) ascii
05      dsh$_locate              (01) ascii
05      dsh$_keep                (01) ascii
05      dsh$_sacount             (02) word
05      dsh$_dam                 (10) ascii
05      dsh$_status              (10) ascii
05      dsh$_create              (08) date (yyyymmdd)
05      dsh$_last                (08) date (yyyymmdd)
05      dsh$_author              (30) ascii
05      dsh$_parameter_length    (02) word
05      dsh$_parameter_block     (200) binary
```

Table 5-1 describes each key in the document header record.

Table 5-1: Description of Document Header Keys

Key	Size	Description
Refnum/ Docnum	Long Word Integer	Reference/Document number (primary key). This field serves two important functions: ● Uniquely identifies every entry within the file cabinet index. ● Orders the file so that the first document reference in any folder is always the most recently created. You can specify a document number in a qualifer for several of the Document Services functions; this allows you to identify a particular document in a file cabinet. However, Document Services does not store the document numbers. Instead, it translates each document number into a reference number. The reference number is a very large number that Document Services stores in the header. This number is a translation of the document number, which you can specify in a qualifier for several of the Document Services functions. Document Services internally translates a document number into a reference number, and then uses the reference number to identify a document. The reason for this internal translation pertains to the way that RMS performs key searches. Document Services never displays reference numbers. The following example illustrates the relationship between a reference number, a document number, and document creation date.

Key	Size	Description
(Continued.)		Refnum / Docnum / Creation-Date: 2^31-5, 5, 5-May-1984; 2^31-4, 4, 4-Apr-1983; 2^31-3, 3, 3-Mar-1982; 2^31-2, 2, 2-Feb-1981; 2^31-1, 1, 1-Jan-1980 Document Services reuses a refnum value only when deleting the most recent document reference in the file cabinet and subsequently creating a new document. A refnum is unique only within its own file cabinet. Also, no duplicates of the refnum key are allowed.
Folder	16 Bytes	Folder name (secondary key). This field specifies the name of the file cabinet folder that contains the referenced document. The value of folder has no impact on the actual location of the document referred to; it is simply a convenience to users and applications. Note that folder values must be in uppercase.
Title	60 Bytes	The user- or application- specified name for the document. The title field consists of two sub-fields. The first is TITLE_KEY, which is a secondary key containing a non-numeric document name. This sub-field allows duplicates. The second sub-field is TITLE_EXT, which contains the remainder of the title; Document Services does not use this portion as a key. Title is an optional field.
Pointer	Long Word Integer	Reserved.
Filnam	40 Bytes	The P/OS filename that Document Services uses to point to the P/OS file containing the text of the desired document.
Keyword	20 Bytes	Document keyword(s). This field contains one or more user-defined keywords that the user can search for in the body of the document. Document Services maintains the field for ALL index entries.

Refnum	Docnum	Creation-Date
2^31-5	5	5-May-1984
2^31-4	4	4-Apr-1983
2^31-3	3	3-Mar-1982
2^31-2	2	2-Feb-1981
2^31-1	1	1-Jan-1980

Key	Size	Description
Locate	1 Byte	Reserved.
Keep	1 Byte	Do not delete file flag. You set this flag to ensure that the P/OS file containing the document text should never be deleted (DOC KILL). Setting keep to Y causes DOC KILL to delete an index entry without deleting the corresponding file. Setting keep to N causes DOC KILL to delete both the index entry <u>and</u> the corresponding file.
Sacount	1 Word Integer	Reserved.
DAM	10 Bytes	Document access method. This field can contain a keyword that identifies a Document Access Method (DAM). A DAM is generally the text editor that the user wants to invoke to edit a particular document. For example, if the user wants to edit a file using the WPS editor keypad, then entering WPS in this field causes Document Services to invoke EDT with the WPS keypad whenever the user edits the document (DOC EDIT). If the you do not specify a value for the DAM field when creating a document, then Document Services assigns the equivalence value of the symbol OA_$_EDITOR. If the symbol OA_$EDITOR is undefined, then Document Services assigns EDT to the DAM field.
(Continued.)		Reserved values for the DAM field are: ● PROSE - Accessible by PROSE editor. ● EDT - Accessible by EDT editor. ● WPS - Accessible by EDT editor with WPS keypad. ● ASCII - Accessible as any ASCII file. By default, same as EDT.

Key	Size	Description
Status	10 Bytes	Reserved.
Create	8 Bytes	Document creation date. This field contains the date that the document header was created. The format is yyyymmdd.
Last	8 Bytes	Last modification date. This field contains the date that the document was last modified. The format is yyyymmdd.
Author	30 Bytes	Document author. This field contains the name of the document's author.
Param Length	2 Bytes	Reserved.
Param Block	200 Bytes	Reserved.

5.1.1 Using Foreign Editors

A foreign editor is an editor that is not supplied with the PRO/Office Workstation kit. You can customize Document Services to use foreign editors, such as Supercomp-20.

To use a foreign editor, first choose a word (10 letters maximum length) that you will use to denote the foreign editor. For example, SUPER could indicate that a document is a Supercomp-20 spreadsheet. Then, use the DOC CREATE or DOC MODIFY command with the /.DAM qualifier to enter this word as the DAM value for your document. Document Services will use this DAM to determine which editor it invokes when accessing the file.

After choosing the DAM name, you must define several dynamic commands. The eqivalence for each command specifies the Flow function to perform when you invoke the DOC command. Note that you cannot use another DOC command within the equivalence for a special dynamic command; this type of recursion is not allowed.

The dynamic commands are:

o **DSI$DAM_damname_EDI**

Document Services attempts to execute this dynamic command whenever you edit (DOC EDIT) the document whose DAM is damname.

- **DSI$DAM_damname_KIL**

  Document Services attempts to execute this dynamic command whenever you attempt to kill (DOC KILL) the document whose dam is damname.

- **DSI$DAM_damname_PRI**

  Document Services attempts to execute this dynamic command whenever you attempt to print (DOC PRINT) the document whose dam is damname.

- **DSI$DAM_damname_DIS**

  Document Services attempts to execute this dynamic command whenever you attempt to display (DOC DISPLAY) the document whose dam is damname.

- **DSI$DAM_damname_CRE**

  Document Services attempts to execute this dynamic command whenever you attempt to create (DOC CREATE) the document with a dam of damname.

For the example using SUPER as the DAM, you would create dynamic commands with the following names:

- DSI$DAM_SUPER_EDI
- DSI$DAM_SUPER_KIL
- DSI$DAM_SUPER_PRI
- DSI$DAM_SUPER_DIS
- DSI$DAM_SUPER_CRE

These commands allow you to define the action Document Services takes for operations on a foreign editor. When DSI handles a document with a foreign DAM, DSI first tries to execute the appropriate dynamic command.

If DSI can't find the appropriate dynamic command definition and the request was to edit the document, then DSI generates an error (dsi$_spawn_error). If the request were to PRINT or DISPLAY, and the appropriate dynamic command isn't defined, then DSI attempts to print or display the document as though it were an ASCII file. KILL would simply destroy the file and document header record. CREATE would create simply create the file.

5.2 ORGANIZATION OF THE DOCUMENT SERVICES TASKS

Document Services consists of two major software components:

- **Document Services Interface (DSI)**

  DSI is a task that serves as the interface between the user and the Document Services functions. When a user invokes a Document Services function, such as DOC EDIT, Flow passes this request to DSI. In turn, DSI might call the Form Interface Facility to display a form requesting user input. Then, to actually perform the Document Services function, DSI calls the other major software component, Callable Document Services.

- **Callable Document Services (CDS)**

  CDS is the task that actually performs a Document Services function. For functions that require manipulation of a document header in a file cabinet, CDS opens the RMS file containing the index, processes header information, and closes the RMS file. In this regard, you can view CDS as an RMS utility program. You can directly access CDS from Flow by invoking the CALL function.

On the application diskettes, the DSI and CDS tasks are called DSI.TSK and CDS.TSK.

As an example of the sequence of events involving DSI and CDS, consider what happens when a user creates and edits a document:

1. From Flow, the user invokes DSI using the DOC CREATE command:

   ```
   > DOC CREATE
   ```

2. DSI calls FI to display a form that prompts the user for the title, keyword, and folder name for the new document.

3. FI displays the form and retrieves the user's input, passing that input back to DSI.

4. DSI assigns the P/OS filename for the new doucment to the symbol OA$CURMES_FILE. Then DSI calls CDS to perform the Document Services CREATE function. This creates the document header in a file cabinet (OA$CABINET, unless specified otherwise). CDS creates a new document number and P/OS filename for the document, then enters the header values in the database index (file cabinet).

5. If the DOC CREATE command specified /EDIT=Y (the default), and the DAM is <u>not</u> foreign, then DSI invokes the DAM to edit the document. If the DAM <u>is</u> foreign, then DSI invokes the dynamic command DSI$DAM_damname_EDI. In both cases, the invoked editor creates the actual P/OS file containing the document text. The editor reads the P/OS filespec from the symbol OA$CURMES_FILE. If no DAM is specified, then Document Services attempts to invoke the editor specified in OA$EDITOR.

Figure 5-2 illustrates the organization of the Document Services Facility, showing DSI, CDS, and the file cabinet. The figure shows that two way communication exists between DSI and CDS, and that CDS has read/write access to the file cabinet. DSI, however, can only read entries in the file cabinet.

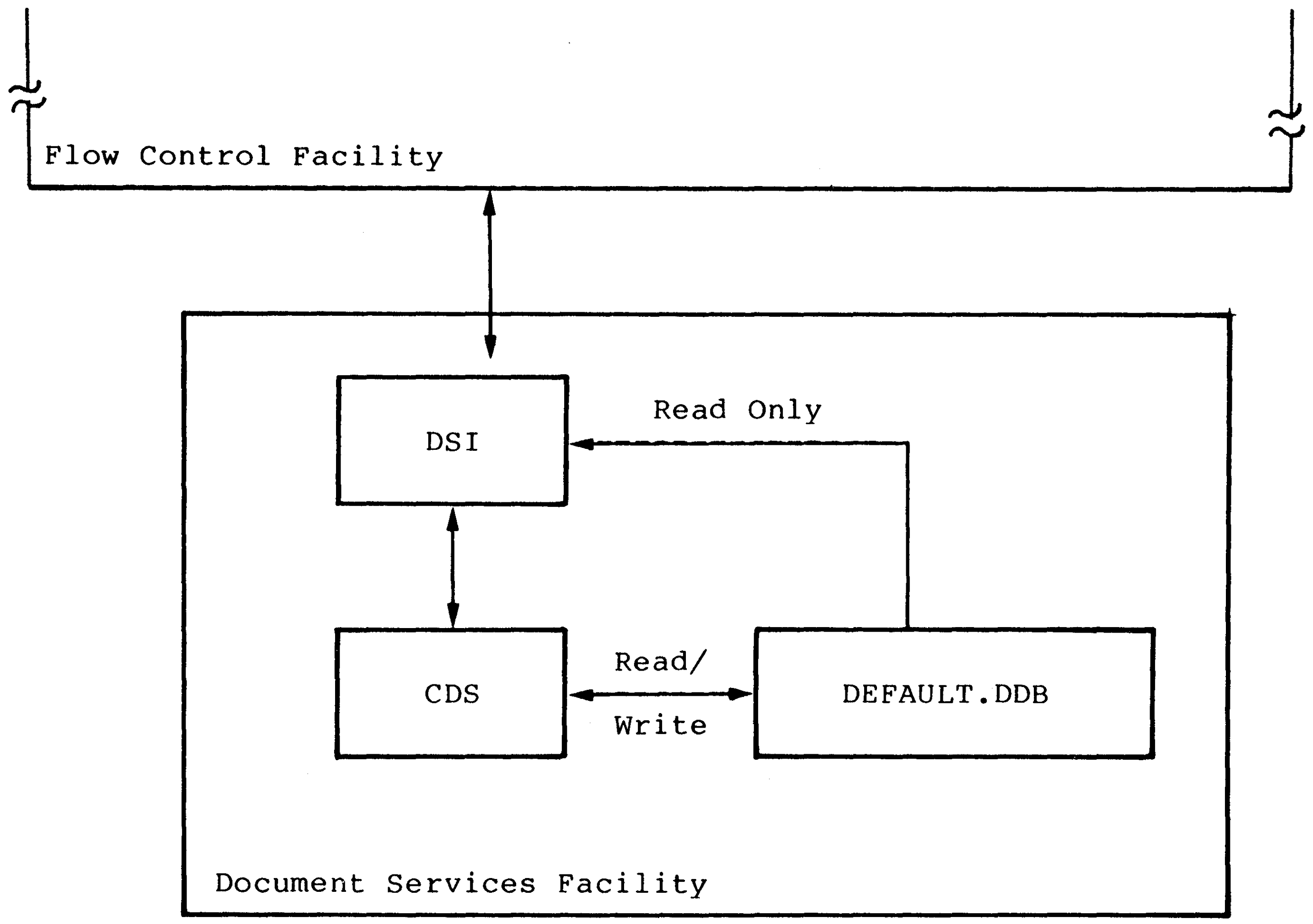


Figure 5-2: Organization of the Document Services Facility

5.3 INVOKING DOCUMENT SERVICES FUNCTIONS

Document Services provides many functions that allow a user to manipulate documents whose headers are stored in a file cabinet. In general, you use the Flow DOC or EXTERNAL functions to access these functions.

The DOC function actually translates into a call to DSI using the EXTERNAL function. For example, the following function invocations are equivalent:

```
> DOC CREATE
> EXTERNAL F$DSI CREATE
```

INVOKING DOCUMENT SERVICES FUNCTIONS

In the EXTERNAL function, the F$ prefix to the DSI taskname indicates that you are calling a task that is accessible only to Flow.

The format of the DOC function is:

```
DOC dsfunction[/qualifier...]
```

Where:

dsfunction	is any of the Document Services functions: ● CAB ● CREATE ● DELETE ● SELECT ● EDIT ● DISPLAY ● PRINT
qualifier	is any qualifier that is valid for the specified operation.

The following sections describe each of the Document Services functions.

NOTE

> You must always use the Flow functions DOC or CALL SWB$DS to access data in the Document Services database. Never attempt such access via your own program. The internal format of the database might change in future versions of the product.

5.3.1 DOC CAB/$REQ=CREATE

Create a file cabinet.

Format

DOC CAB/$REQ=CREATE[/qualifier...]

Description

The DOC CAB function allows you to perform operations on the file cabinet itself, rather than on headers within a file cabinet. DOC CAB/$REQ=CREATE creates a new file cabinet.

The qualifiers are:

/$CAB=cabname

> The /$CAB qualifier specifies the P/OS filename of the cabinet you are creating. The cabname you specify can be a complete file specification for the file cabinet.
>
> Document Services requires the cabinet name in order to create the cabinet. Consequently, if you do not specify /$CAB, Document Services displays a form to retrieve the cabinet name (unless you also specify /FORM=N, in which case you receive an error).

/$TIT=title

> The /$TIT qualifier specifies the title of the cabinet you are creating.
>
> Document Services requires a title in order to create the cabinet. Consequently, if you do not specify /$TIT, Document Services displays a form to retrieve the title (unless you also specify /FORM=N, in which case you receive an error).

/FORM=Y or N

> The /FORM qualifier specifies whether or not Document Services should request FI to display a form that prompts the user for the filename and title of the new cabinet.
>
> By default, /FORM=Y. When /FORM=Y, Document Services <u>always</u> requests FI to display the form (a filename or title you specify appears as a default value on the form). If you specify /FORM=N to suppress the form display, you must also specify values for /$CAB and /$TIT, or you receive an error message.

5.3.2 DOC CAB/$REQ=DELETE

Delete a file cabinet.

Format

```
DOC CAB/$REQ=DELETE[/qualifier...]
```

Description

The DOC CAB function allows you to perform operations on the file cabinet itself, rather than on headers within a file cabinet. DOC CAB/$REQ=DELETE destroys all headers in an existing file cabinet, kills all document files that the cabinet's headers refer to (see DOC KILL), and then destroys the file cabinet itself. The file cabinet and its headers cannot be recovered.

The qualifiers are:

/$CAB=cabname

> The /$CAB qualifier specifies the P/OS filename of the cabinet you are deleting. The cabname you specify can be a complete file specification for the file cabinet.
>
> If you do not specify /$CAB, Document Services attempts to delete the current file cabinet.

5.3.3 DOC CAB/$REQ=SELECT

Select a file cabinet.

Format

```
DOC CAB/$REQ=SELECT
```

Description

The DOC CAB function allows you to perform operations on the file cabinet itself, rather than on headers within a file cabinet. DOC CAB/$REQ=SELECT allows you to specify a new current file cabinet.

When you invoke this function, Document Services requests FI to display a that lists all the file cabinets in ascending alphabetic order, and allows the user to select one.

The form can have multiple screens. If the user presses the MAIN SCREEN key while viewing the form, FI displays the first screen of the form. If the user presses the EXIT key, Document Services returns without performing the operation.

When you select a new current file cabinet, Document Services assigns the P/OS filespec of the new current cabinet to the symbol OA$CABINET and the title of the cabinet to OA$CABINET_TITLE. Document Services assigns null to all other Document Services symbols whose values depend on the current file cabinet:

- OA$CURMES_DAM
- OA$CURMES_FILE
- OA$CURMES_FOLDER
- OA$CURMES_NBR
- OA$CURMES_TITLE

Example

```
> DOC CAB/$REQ=SELECT
```

5.3.4 DOC CREATE

Create a header in the specified cabinet.

Format

DOC CREATE[/qualifier...]

Description

This function creates only a document's header; it does not create the actual P/OS file. Document Services lets the editor create the P/OS file the first time you edit the document.

Many of the qualifiers allow you to specify values for fields in the new header (see the description of the header fields in Table 5-1).

The qualifiers are:

/.FOL=folname

> The /.FOL qualifier specifies the name of the folder in which you place the new document.
>
> Document Services requires a folder name in order to create the document header. Consequently, if you do not specify /.FOL, Document Services displays a form to retrieve the folder name (unless you also specify /FORM=N, in which case you receive an error).

/.TIT=title

> The /.TIT qualifier specifies the title of the document you are creating.
>
> Document Services requires a title in order to create the document header. Consequently, if you do not specify /.TIT, Document Services displays a form to retrieve the title (unless you also specify /FORM=N, in which case you receive an error).

/.FIL=filname

The /.FIL qualifier specifies the RMS filename of the new document. This qualifier is optional.

/FORM=Y or N

The /FORM qualifier specifies whether or not Document Services should request FI to display a form that prompts the user for the cabinet, title, keywords, and folder for the new document.

By default, /FORM=Y. When /FORM=Y, Document Services always requests FI to display the form (a filename or title you specify appears as a default value on the form). If you specify /FORM=N to suppress the form display, you must also specify values for /.TIT, /.FOL, and /.KEY, or you receive an error message.

/.KEY=keywords

This qualifier allows you to enter keywords in the keyword field of the header. By default, this field is blank. See Table 5-1 for details on this field.

/.KEP=Y or N

This qualifier allows you to enter a Y(es) or N(o) value in the keep field of the header. By default, /.KEP=N. See Table 5-1 for details on this field.

/.DAM=dam

This qualifier allows you to enter a value in the Document Access Method (DAM) field in the header. By default, /.DAM="ASCII". See Table 5-1 for details on this field.

/.STA=status

This qualifier allows you to enter a value in the status field of the header. By default this field is blank. See Table 5-1 for details on this field.

/.AUT=author

This qualifier allows you to enter a value in the author field of the header. By default this field is blank. See Table 5-1 for details on this field.

/$CAB=cabname

The /$CAB qualifier specifies the P/OS filename of the cabinet in which you are creating the header. The cabname you specify can be a complete file specification for the file cabinet. If you do not specify /$CAB, Document Services attempts to create the header in the current file cabinet.

/EDIT=Y or N

This qualifier allows you to specify whether or not the user will edit the document after having created its header. By default, /EDIT=Y.

5.3.5 DOC DELETE

Place a document's header in the folder WASTEBASKET.

Format

```
DOC DELETE[/qualifier...]
```

Description

Deleting a document doesn't erase that document. Instead, deleting a document merely places its header in the WASTEBASKET folder. Once its header is in the WASTEBASKET folder, a document can be erased by "emptying the wastebasket" (see DOC KILL).

The qualifiers are:

/$CAB=cabname

> If you specify /$CAB, you must also specify /$NUM. The /$CAB qualifier sets the current file cabinet to cabname before performing the operation. The cabname you specify must be the name of an existing file cabinet. You can provide a complete file specification for the file cabinet. If you do not specify /$CAB, then Document Services uses either the current file cabinet (as defined by the symbol OA$CABINET), or the document specified by /$NUM in the current file cabinet.

/$NUM=docnum

> This qualifier allows you to specify the document number of the document on which you perform the operation. The number you specify, docnum, must refer to an existing document number in either the current file cabinet, or in the file cabinet specified in /$CAB.

Examples

```
> DOC DELETE/$CAB=MYCAB.DDB/$NUM=45
> DOC DELETE/$NUM=45
> DOC DEL/$CAB=CABBY.DDB/$NUM=8
```

5.3.6 DOC DISPLAY

Show a document on the terminal screen.

Format

```
DOC DISPLAY[/qualifier...]
```

Description

The DISPLAY function invokes the TYPE utility, which is the utility that Flow's TYPE function uses. Before showing the file, DISPLAY clears the screen.

The qualifiers are:

/$CAB=cabname

> If you specify /$CAB, you must also specify /$NUM. The /$CAB qualifier sets the current file cabinet to cabname before performing the operation. The cabname you specify must be the name of an existing file cabinet. You can provide a complete file specification for the file cabinet. If you do not specify /$CAB, then Document Services uses either the current file cabinet (as defined by the symbol OA$CABINET), or the document specified by /$NUM in the current file cabinet.

/$NUM=docnum

> This qualifier allows you to specify the document number of the document on which you perform the operation. The number you specify, docnum, must refer to an existing document number in either the current file cabinet, or in the file cabinet specified in /$CAB.

Examples

```
> DOC DISPLAY/$CAB=MYCAB.DDB/$NUM=45
> DOC DIS/$NUM=18
> DOC DISPLAY
```

5.3.7 DOC EDIT

Invoke the editor indicated in the DAM field of a document's header.

Format

```
DOC EDIT[/qualifier...]
```

Description

The qualifiers are:

/$CAB=cabname

If you specify /$CAB, you must also specify /$NUM. The /$CAB qualifier sets the current file cabinet to cabname before performing the operation. The cabname you specify must be the name of an existing file cabinet. You can provide a complete file specification for the file cabinet. If you do not specify /$CAB, then Document Services uses either the current file cabinet (as defined by the symbol OA$CABINET), or the document specified by /$NUM in the current file cabinet.

/$NUM=docnum

This qualifier allows you to specify the document number of the document on which you perform the operation. The number you specify, docnum, must refer to an existing document number in either the current file cabinet, or in the file cabinet specified in /$CAB.

/.DAM=dam

This qualifier allows you to override the value in the DAM field in the header. By default, Document Services uses the the value stored in the DAM field. See Table 5-1 for details on the possible values for this qualifier.

Examples

```
> DOC EDIT/$CAB=MYCAB.DDB/$NUM=45
> DOC EDIT
> DOC EDIT/$NUM=34       ! Document 34 in current cabinet
```

5.3.8 DOC KILL

Erase a document header and document file.

Format

```
DOC KILL[/qualifier...]
```

Description

This operation is a companion to DOC DELETE. Where DOC DELETE moves a document header into the WASTEBASKET folder to await final deletion, KILL immediately erases the header and document text file.

Note the effect of the keep field (/.KEP) on the operation of this functions:

- If the value in the keep field is N, then DOC KILL immediately erases the header and document text file.
- If the value in the keep field is Y, then DOC KILL erases only the header, leaving the document text file intact.

The qualifiers are:

/$CAB=cabname

If you specify /$CAB, you must also specify either /$NUM or /.FOL. The /$CAB qualifier sets the current file cabinet to cabname before performing the operation. The cabname you specify must be the name of an existing file cabinet. You can provide a complete file specification for the file cabinet. If you specify neither /$CAB nor /.FOL, then Document Services kills one of the following:

- The current document in the current file cabinet (as defined by the symbol OA$CABINET).
- A document specified by /$NUM in the current file cabinet.

/$NUM=docnum

This qualifier allows you to specify the document number of the document on which you perform the operation. The number you specify, docnum, must refer to an existing document number in either the current file cabinet, or in the file cabinet specified in /$CAB.

/.FOL=folname

This qualifier allows you to kill all the documents in a specified folder. The value of folname must be the name of an existing folder in either the current cabinet (if you omit /$CAB) or in the cabinet specified by /$CAB.

Examples

```
> DOC KILL/$CAB=CAB6.DDB/$NUM=12
> DOC KILL/$NUM=34
> DOC KILL/.FOL=WASTEBASKET      ! Empty the WASTEBASKET
```

5.3.9 DOC MODIFY

Display and modify selected fields in a document header.

Format

DOC MODIFY[/qualifier...]

Description

See Table 5-1 for details on the fields in the header that you can modify.

The qualifiers are:

/$CAB=cabname

If you specify /$CAB, you must also specify /$NUM. The /$CAB qualifier sets the current file cabinet to cabname before performing the operation. The cabname you specify must be the name of an existing file cabinet. You can provide a complete file specification for the file cabinet. If you do not specify /$CAB, then Document Services uses either the current file cabinet (as defined by the symbol OA$CABINET), or the document specified by /$NUM in the current file cabinet.

/FORM=Y or N

The /FORM qualifier specifies whether or not Document Services should request FI to display a form that prompts the user for values of fields in the header, showing those specified with qualifiers as defaults. By default, /FORM=Y.

/$NUM=docnum

This qualifier allows you to specify the document number of the document on which you perform the operation. The number you specify, docnum, must refer to an existing document number in either the current file cabinet, or in the file cabinet specified in /$CAB.

/.FOL=folname

The /.FOL qualifier allows you to modify the name of the folder in which you store the document.

/.TIT=title

The /.TIT qualifier allows you to modify the title of the document.

/.FIL=filname

The /.FIL qualifier allows you to modify the RMS filename of the document.

/.KEP=Y or N

This qualifier allows you to modify the Y(es) or N(o) value in the keep field of the header.

/.DAM=dam

This qualifier allows you to modify a value in the Document Access Method (DAM) field in the header.

/.KEY=keywords

This qualifier allows you to modify keywords in the keyword field of the header.

/.STA=status

This qualifier allows you to modify a value in the status field of the header.

/.AUT=author

This qualifier allows you to modify a value in the author field of the header.

5.3.10 DOC PRINT

Queue a print request to the printer.

Format

```
DOC PRINT[/qualifier...]
```

Description

Use this command to perform local printing, that is, to print a document at the printer connected directly to your Professional. For information on remote printing (printing a document on your host), see Chapter 8, Section 8.5.

The qualifiers are:

/$CAB=cabname

> If you specify /$CAB, you must also specify /$NUM. The /$CAB qualifier sets the current file cabinet to cabname before performing the operation. The cabname you specify must be the name of an existing file cabinet. You can provide a complete file specification for the file cabinet. If you do not specify /$CAB, then Document Services uses either the current file cabinet (as defined by the symbol OA$CABINET), or the document specified by /$NUM in the current file cabinet.

/$NUM=docnum

> This qualifier allows you to specify the document number of the document on which you perform the operation. The number you specify, docnum, must refer to an existing document number in either the current file cabinet, or in the file cabinet specified in /$CAB.

5.3.11 DOC SELECT

Specify the current document.

Format

```
> DOC SELECT/BY=selcriterion[/qualifier...]
```

Where:

selcriterion	can be NUM, FOL, or TIT. See the qualifer descriptions below for details.

Description

A user can perform other Document Services functions only on the current document, whatever that document is. If the current document is not the one desired, the user can SELECT a document with this function, thereby making the specified document the current one.

The SELECT function calls the Form Interface to display a form that presents several documents and allows the user to choose one. You can specify selection criteria with the /BY qualifier. See the description of the qualifiers below.

The form can have multiple screens. To display the first screen of the form, the user can press the MAIN SCREEN key. To exit from the SELECT function without updating the current file cabinet, the user can press the EXIT key.

You must specify the /BY qualifer. Descriptions of qualifiers follow.

/BY=selcriterion

This qualifier specifies the selection criteria that Document Services uses when reading records for the form. You must specify one of the following values for selcriterion:

- NUM -- requests that Document Services display the documents in reverse chronological order of creation. The form shows the document number, followed by a portion of the title. Document Services ignores folders.

- FOL -- requests that Document Services display the documents by folder name. The form shows the folder names in ascending alphabetic order.

- TIT -- requests that Document Services display the documents by title, in ascending alphabetic order.

/$CAB=cabname

The /$CAB qualifier sets the current file cabinet to cabname before performing the operation. The cabname you specify must be the name of an existing file cabinet. You can provide a complete file specification for the file cabinet. If you do not specify /$CAB, then Document Services uses the current file cabinet, as defined by the symbol OA$CABINET.

/.FOL=folname

This qualifier allows you to specify the folder from which the user can select a document. Document Services displays only the documents in the specified folder.

Examples

```
> DOC SEL/BY=NUM/$CAB=MYCAB.DDB
> DOC SEL/BY=FOL/$CAB=[CABINETS]CAB1.DDB
> DOC SEL/BY=FOL
```

5.3.12 DOC foreigncmd

Execute a Document Services foreign command.

Format

```
DOC foreigncmd
```

Where:

foreigncmd	is the name of your foreign command, as specified as part of the symbol name DSI$COM_foreigncmd.

Description

A Document Services foreign command is a command whose action you have defined in a symbol equivalence. You define the action of the foreign command in the equivalence of the symbol DSI$COM_foreigncmd.

Foreign commands are a means for you to formulate your own commands to handle documents.

There is no relationship between a foreign command and a foreign DAM, except that you would likely want to define a set of foreign commands for every foreign DAM that you use.

Example

Suppose you want to define the foreign command DOC XXX, which would allow you to perform the XXX operation on a document. Your first step is to define the symbol DSI$COM_foreigncmd:

```
> DEFINE/SYMBOL/SYSTEM  DSI$COM_XXX "@[COMMANDS]XXX"
```

Then you would create your command procedure [COMMANDS]XXX.COM. This file can contain the actual Flow functions that Flow would perform whenever you invoke the foreign command. Note that you cannot invoke the DOC function as part of a foreign command.

You would invoke the foreign command XXX as follows:

```
> DOC XXX
```

Here's a more specific example. Suppose you want to put .TSK files in a separate cabinet, and define a foreign command DOC RUN. Define the foreign command as follows:

```
> DEFINE/SYMBOL/SYSTEM  DSI$COM_RUN  "RUN OA$CURMES_FILE"
```

5.4 DOCUMENT SERVICES SYMBOLS

Table 5-2 describes all the symbols that Document Services uses.

Table 5-2: Symbols Used by Document Services

Symbol	Description
DSI$COM_fcmd	The symbol in which you store the associated operation for a foreign command named fcmd.
OA$_CABINET	The P/OS file specification of the current file cabinet.
OA$_CABINET_TITLE	The title of the current file cabinet.
OA$_CURMES_DAM	The DAM of the current document.
OA$_CURMES_FILE	The P/OS file specification of the current document.
OA$_CURMES_FOLDER	The current folder.
OA$_CURMES_NUMBER	The document number of the current document.
OA$_CURMES_TITLE	The document title of the current document.
OA$_EDITOR	The current preferred editor.

CHAPTER 6

USING THE SYMBOL SERVICES FACILITY

A symbol is a combination of a name and an equivalence value. You define a symbol by associating a name with an equivalence value in any of Flow's symbol-defining functions. Tags, dynamic commands, key definitions, and application definitions are all symbols, albeit special kinds of symbols.

The Symbol Services Facility ("Symbol Services") creates, deletes, and translates symbols, and stores their definitions in a symbol database. This chapter describes the symbol database, the tasks that compose Symbol Services, and the qualifiers that you can specify when performing operations on symbols.

6.1 ORGANIZATION OF THE SYMBOL SERVICES DATABASE

A symbol database consists of a collection of symbol tables. In PRO/Office Workstation, a symbol table is an RMS index file containing symbol definitions.

Symbol Services provides three kinds of symbol tables:

- **Process Table**

  This kind of table contains symbol definitions used for tasks (we use the terms "process" and "task" interchangeably). All tasks share a single process table.

  The P/OS filename for the process table is DW1:[ZZOASYM]PRCPROCES.SYM.

- **User Table**

  This table stores symbols that contain information relevant to the Workstation user. Examples of this kind of information are the user's name, node, telephone number, title, and department.

  The P/OS filename for the user table is DW1:[ZZOASYM]USRUSER.SYM

- **System Table**

  The system table contains symbol definitions that are valid throughout a particular installation of PRO/Office Workstation. These definitions are "global" in the sense that any task or user on a particular system can access them. Symbol Services creates one system table for a system.

  The P/OS filename for the system table is DW1:[ZZOASYM]SYSSYSTEM.SYM.

Note that Symbol Services places all the tables in the directory [ZZOASYM].

6.2 DEFINING SYMBOLS

You define a symbol by invoking any of Flow's <u>symbol-defining</u> functions:

- DEFINE/APPLICATION
- DEFINE/COMMAND
- DEFINE/KEY
- DEFINE/SYMBOL
- DEFINE/TAG
- INQUIRE
- LET
- FIELD

DEFINING SYMBOLS

Both the name and equivalence contain a string of ASCII characters. The maximum name length is 32 bytes; the maximum equivalence length is 512 bytes. Symbol Services translates the symbol name into uppercase ASCII characters without any blank, tab, or carriage return characters.

You can optionally specify a valid abbreviation for a symbol name when you define the symbol. To do this, you provide the full symbol name with an asterisk (*) inserted at the position representing your desired minimal abbreviation. Also, you must specify the /OVERRIDE qualifier. For example:

```
> DEFINE/SYMBOL/OVERRIDE CON*NECT  "@[COMMANDS]CONNECT"
```

Given the symbol CONNECT defined above, it is sufficient to specify the three characters CON to fully identify the symbol.

Using symbol abbreviations can cause ambiguous symbol definitions, which can be dangerous. This is because Symbol Services deletes ambiguous definitions from the process table. You might unintentionally delete one or more symbol definitions.

For example, assume you have defined a set of symbols whose names begin with the same character (let's use "A"). Then, suppose you subsequently define a symbol with an allowed minimum abbreviation of just one character (for example, "A*BLE"). In this case, defining A*BLE causes the complete set of symbols whose names start with "A" to become ambiguous -- hence Symbol Services deletes all of them.

You can specify several symbol qualifiers when you define a symbol. The qualifiers are:

- /NODELETE
- /OVERRIDE
- /PROCESS
- /SYSTEM
- /USER
- /VOLATILE

NOTE

Version 1.0 of PRO/Office Workstation does not allow you to specify these qualifiers on the LET or FIELD functions.

Table 6-1 shows valid paired combinations of these qualifiers. At the intersection of a row and column, "Yes" or "No" indicates whether or not you can specify the associated pair of qualifiers in a particular invocation of a symbol-defining function.

Table 6-1: Valid Combinations of Symbol Qualifiers

Qualifier	/NODEL	/OVERR	/PROCE	/SYSTE	/USER	/VOLAT
/NODEL	-	Yes	Yes	Yes	Yes	Yes
/OVERR	No	-	Yes	Yes	Yes	Yes
/PROCE	Yes	Yes	-	No	No	Yes
/SYSTEM	Yes	Yes	No	-	No	Yes
/USER	Yes	Yes	No	No	-	Yes
/VOLAT	No	No	Yes	Yes	Yes	-

The following sections describe the qualifiers.

6.2.1 /NODELETE

The NODELETE attribute prohibits Symbol Services from deleting or redefining the symbol unless you specify the /OVERRIDE qualifier on the DELETE or a symbol-defining function.

If you attempt to delete or redefine a NODELETE symbol in the table in which it is defined as NODELETE, without specifying the /OVERRIDE qualifier, Symbol Services returns an error message.

Why Use It?

You might use certain symbols within your customized Workstation that the user should never be able to delete. For example, suppose you define a symbol as follows:

```
> DEFINE/SYMBOL/SYSTEM/NODELETE GM$_DIVISION  "BUICK"
```

Such a symbol might be a field on various forms that your customized Workstation displays. The symbol cannot be redefined or deleted without explicitly specifying the /OVERRIDE qualifer:

```
> DEFINE/SYMBOL/SYSTEM/OVERRIDE GM$_DIVISION  "CHEVROLET"
> DELETE/SYMBOL/SYSTEM/OVERRIDE GM$_DIVISION
```

6.2.2 /OVERRIDE

This qualifier allows you to override /NODELETE attribute of a symbol. It also lets you define a minimum abbreviation for a symbol.

Why Use It?

Use this qualifier to redefine a symbol that you have previously defined with the /NODELETE qualifier. You receive an error if you try to delete such a symbol without specifying /OVERRIDE.

Also use /OVERRIDE to define a symbol with a minimum abbreviation. You indicate the minimum abbreviation by placing an asterisk just after the final character in the substring of the symbol name.

For example, the abbreviation MYN in the following symbol definition adequately represents the symbol MYNAME:

```
> DEFINE/SYMBOL/OVERRIDE  MYN*AME  "TIM OCONNOR"
```

Section 6.2.1 gives an example of /NODELETE and /OVERRIDE.

6.2.3 /PROCESS

The /PROCESS qualifier forces Symbol Services to place the specified definition in the process table. By default, if you do not specify any of the qualifiers /PROCESS, /USER, /SYSTEM, or /VOLATILE, Symbol Services places the definition in the process table.

You can simultaneously define a symbol in a process table and either a user table or the system table. However, unless you explicitly specify the user or system table when you redefine or delete a symbol, Symbol Services by default performs the operation on the symbol defined in the process table.

See Section 6.4 for a description of the order in which Symbol Services searches the tables to find a definition.

Why Use It?

You should obtain slightly greater system performance if your symbols are defined in the process table rather than the system table.

a user table for a single-user Workstation, or the system table for a muliple-user Workstation.

The following examples are equivalent:

```
> DEFINE/SYMBOL/PROCESS  hello  "COMMAND [COMMANDS]HELLO"
> DEFINE/SYMBOL hello "COMMAND [COMMANDS]HELLO"
```

6.2.4 /SYSTEM

The /SYSTEM qualifier forces Symbol Services to place the specified symbol definition in the system table.

See Section 6.4 for a description of the order in which Symbol Services searches the tables to find a definition.

Why Use It?

Put symbols that you do not frequently access in the system table. Place those that you do frequently access in the process table, for better performance.

6.2.5 /USER

The /USER qualifier forces Symbol Services to place the specified definition in the user table. The user, and all tasks spawned from the user's initial Flow task, have complete access to definitions located in the user table.

Why Use It?

The user table is a good place to store user-related information, such as name, phone number, and badge number.

6.2.6 /VOLATILE

The VOLATILE attribute prohibits Symbol Services from storing a symbol in the process table on disk, where it would otherwise reside. Instead, a volatile symbol exists only in main memory. Consequently, when you delete a volatile symbol, a definition of the symbol residing in the symbol database (in any table) becomes active again.

You delete all volatile symbols defined by a task when you turn off the Professional. Alternatively, you can use the DELETE/SYMBOL function to delete a particular volatile symbol.

If Symbol Services finds no more room in the dynamic memory pool when you define a volatile symbol, you receive a create error.

Why Use It?

Volatile symbols provide the highest performance; they are always in memory and do not have to be read from disk.

Also, the VOLATILE attribute allows you to temporarily redefine an existing symbol. For example, suppose you normally have defined key F17 as follows:

```
> DEFINE/KEY 17 "RUN DTE"
```

Symbol Services places this key definition in the process table.

Now assume that you would temporarily like to use key 17 to run a command procedure that you are testing. You can define the key as a volatile symbol:

```
> DEFINE/KEY/VOLATILE 17 "COMMAND TESTPROC"
```

When you subsequently remove this volatile key definition, you automatically regain the original definition ("RUN DTE"). To remove the volatile definition:

- Invoke the DELETE/KEY function.
- Turn off the Professional.

6.3 DELETING SYMBOLS

You explicitly delete a symbol by invoking any the Flow DELETE functions:

- DELETE/APPLICATION
- DELETE/COMMAND
- DELETE/KEY
- DELETE/SYMBOL
- DELETE/TAG

The DELETE functions remove a symbol definition from a specified symbol table. If no table is specified, Symbol Services attempts to delete the symbol from the process table.

Table 6-2 describes the qualifiers you can specify for the DELETE functions.

Table 6-2: Qualifiers for Deleting a Symbol

Qualifier	Description
/PROCESS	Deletes a symbol definition from the process table only. This is the default.
/SYSTEM	Deletes a symbol definition from the system table only.
/USER	Deletes a symbol from the user table only.
/OVERRIDE	Allows you to delete a symbol defined with /NODELETE.

6.4 REFERRING TO SYMBOLS

Flow passes all parsed symbol references to Symbol Services for interpretation. Symbol Services attempts to translate a symbol by searching the tables for its definition. The following list shows the order, beginning from the top and ending at the bottom, in which Symbol Services searches for a definition. Note that Symbol Services maintains a symbol cache for improved performance.

REFERRING TO SYMBOLS

1. Process Table:
    a. Volatile symbol
    b. Cached symbol
    c. Nonvolatile symbol

2. User Table:
    a. Volatile symbol
    b. Cached symbol
    c. Nonvolatile symbol

3. System Table:
    a. Volatile symbol
    b. Cached symbol
    c. Nonvolatile symbol

If Symbol Services finds the appropriate definition at any point in the search, it stops the search and uses the definition located. Otherwise, the Facility returns an error.

CHAPTER 7

USING THE MAIL SERVICES FACILITY

The Mail Services Facility ("Mail Services") uses the databases of two other facilities: Document Services (to store, retrieve, and modify messages), and Network Services (to store path records of message recipients). The Mail Services Facility has its own functions that you can invoke by using the Flow function MAIL.

This chapter describes the Mail Services databases, the organization of the Mail Services Tasks, and the functions that allow you to perform Mail operations.

7.1 ORGANIZATION OF THE MAIL SERVICES DATABASE

Mail Services uses several folders in the Document Services database; these are all in cabinet [ZZDOC0]DEFAULT.DDB, and are called:

- READ

  Contains messages received from the host computer that the user has read.

- UNREAD

  Contains messages received from the host computer that the user has not read.

- CREATED

  Contains messages that have been created, but not yet queued for sending to the host computer.

- PENDING_PICKUP

  Contains messages that are queued for sending to the host computer. You can remove a message from this folder to avoid its being sent. See Section 7.3.7 for details.

- SENT

  Contains messages that Network Services has sent to the host computer.

- DEAD_MAIL

  Contains messages that Network Services did not send to the host due to errors in attempted transmission. These errors generally involve the format of the message.

In addition to these folders, Mail Services uses these files:

- [ZZDOC0]FMTMPLT.DAT

  A template file containing a heading for all mail messages. You can indicate symbols to be translated in the displayed text by surrounding the symbol names with angle brackets.

- [ZZDOC0]FMUSER.DAT

  A validation file containing a list of valid addressees. This file is maintained by the DTF task in Mail Services.

7.2 ORGANIZATION OF THE MAIL SERVICES TASKS

The Mail Services Facility consists of three major software modules:

- **Flow Mail (FMAIL)**

  FMAIL is a task that serves as the interface between the user and Mail operations. FMAIL creates messages by calling Document Services routines via DSI. Also, FMAIL reads messages that the host VAX transmitted to the Professional, in the folder UNREAD. Finally, FMAIL marks messages for sending by placing them in folder PENDING_PICKUP. (Note that changing a document's folder merely involves changing the folder field in its document header.) Once a message is in the folder PENDING_PICKUP, it can be sent to the host VAX by Mail's other module, TMAIL.

- **Transport-Level Mail (TMAIL)**

  TMAIL, in combination with the Network Services, is the task that actually sends and receives messages. TMAIL periodically reads messages from the folder PENDING_PICKUP and sends them to the host VAX, using Network Services functions. TMAIL places messages successfully sent in the folder SENT; it places messages not successfully sent in the folder DEAD_MAIL. Finally, FMAIL receives messages from the host VAX and places them in the folder UNREAD. You can use the MAIL BACKGROUND function to set the wakeup interval for TMAIL.

- **DTF**

  DTF is a task that maintains a list of valid addressees in a validation file. This is an RMS indexed file located in [ZZDOCO]FMUSER.DAT. When you SEND, FORWARD, or ANSWER a message, FMAIL checks the users you enter in the TO: and CC: fields against users stored in the validation file. Note that the validation file provides a generic search capability, thus allowing you to specify substrings of your valid addressees in the TO: and CC: fields. TMAIL does not send a message unless all addressees are valid. You can invoke DTF as follows to to perform maintenance on the validation file:

  ```
  > INSTALL [ZZFLOW]DTF.TSK
  > CALL DTF/$FORM=FMUSER/$LIB=OAFMS.FLB
  ```

As an example of the sequence of events involving FMAIL and TMAIL, consider what happens when a user creates and sends a mail message:

1. From Flow, the user invokes FMAIL using the MAIL (or EXTERNAL) command. In command mode:

   ```
   > MAIL CREATE
   ```

2. FMAIL calls FI to display a form that prompts the user to fill in the TO:, CC:, and SUBJECT: fields of the message heading.

3. FMAIL reads the validation file [ZZDOCO]FMUSER.DAT to validate the adressees specified in the TO: and CC: fields.

4. After the user enters data in the SUBJECT: field, FMAIL calls Document Services to create a document. In this case, the P/OS filename for the document has the extension .MSG -- rather than .OAD --- to distinguish this document as a mail message. The call to Document Services invokes the preferred editor (symbol OA$EDITOR), and then places the new message in the folder CREATED.

5. The user then wants to send the message, and invokes FMAIL from Flow with the SEND function:

   ```
   > MAIL SEND
   ```

6. FMAIL moves the current message from folder CREATED to folder PENDING_PICKUP.

7. At a predetermined interval, TMAIL activates, reading the message from the PENDING_PICKUP folder. The user can specify a wakeup interval for TMAIL via the Electronic Mail Setup menu, which invokes the following Flow command:

   ```
   > MAIL BACKGROUND/ENABLE=interval
   ```

   See Section 7.3.2 for details.

8. TMAIL invokes the Network Services task XCOM, using the user's stored path record to login to the host VAX. Once logged on to the VAX, TMAIL initiates the command procedures PROA1.COM (start VAX ALL-IN-1) and PROSEND.COM (send mail from PRO to VAX ALL-IN-1 and restart TMAIL on the Professional).

9. TMAIL transfers the message from the PENDING_PICKUP folder to the SENT folder.

10. TMAIL then "hibernates" until its next wakeup period arrives.

Figure 7-1 illustrates the organization of the Mail Services tasks, showing FMAIL and TMAIL, as well as the relationship to Document Services and Communication Services. The foreground area consists of those tasks that that can interact directly with the user -- FMAIL and DSI.

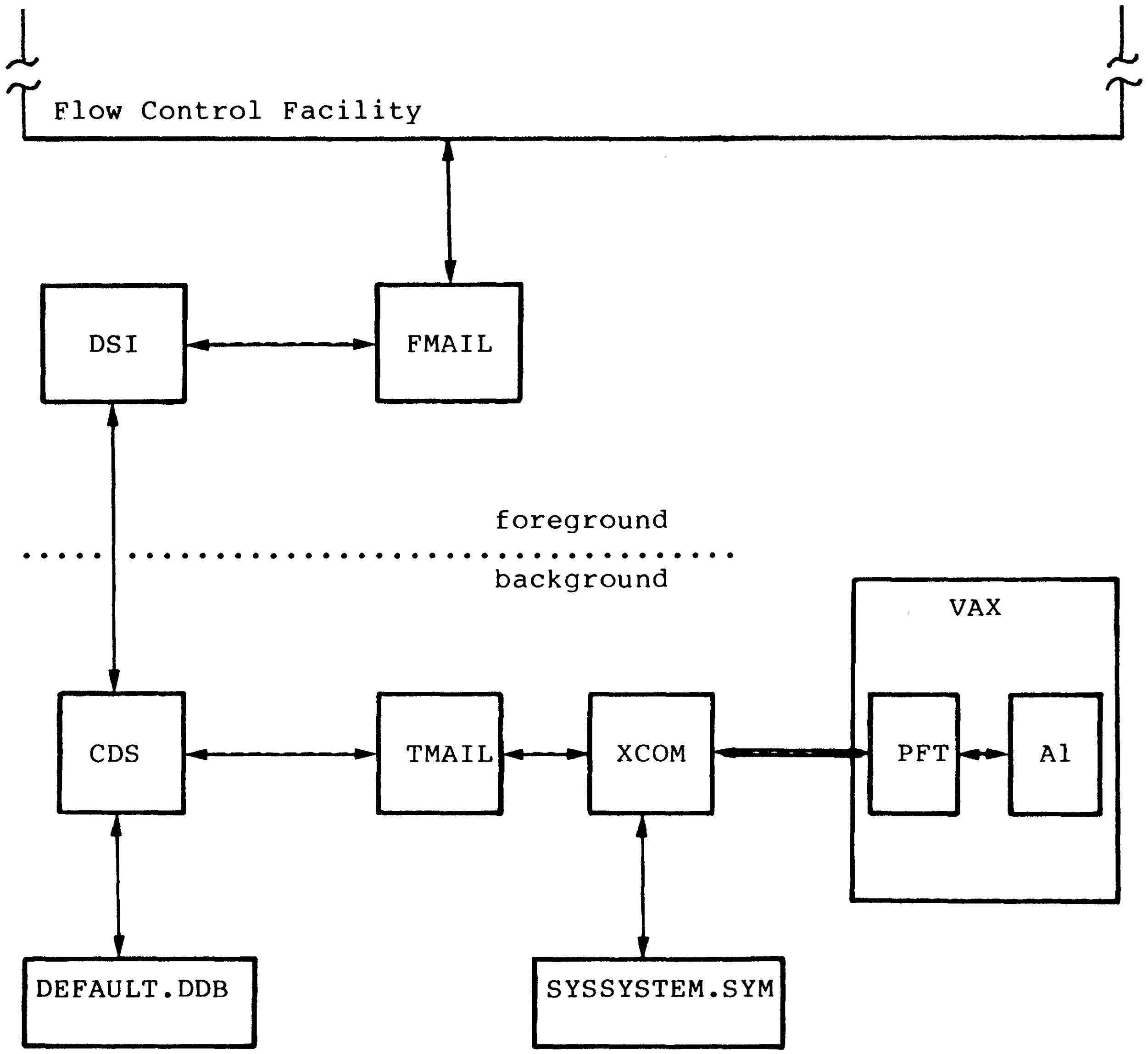


Figure 7-1: Organization of the Mail Services Facility

7.2.1 VAX Command Procedures Used by Mail Services

PRO/Office Workstation uses a number of command procedures to communicate with VAX ALL-IN-1. Mail Services uses a subset of these command procedures to send and receive mail messages:

- PROA1.COM
- PROSEND.COM
- PROREAD.COM
- PROMEXT.COM
- PROMEXT.DTR

For details on these command procedures, see Chapter 8, section 8.6.1.

7.2.1.1 Path Record Required by TMAIL - In order for TMAIL to correctly send and receive mail using the command procedures, you must define a path record named:

```
MAIL::*
```

To define this path record, do either of the following:

- Display the form XNETSU by invoking the Flow command

  ```
  > MENU XNETSU
  ```

  Then select the menu option CREATE.

- Invoke XLIB CREATE as described in Chapter 8, Section 8.3.

Enter the following values for the "Select a library record" portion of the XLIB session:

```
Select a library record

Enter Node?     MAIL
Enter Object?   *
```

Values you enter in the "Create a path record" portion of the XLIB session depend on your connection to the host VAX (for example, a hardwired or Gandalf connection). See Chapter 8 for details. A sample session for a hardwired connection follows.

```
Create a path record

        FAC_CODE?  2
        FAC_NODE?  <CR>
      FAC_OBJECT?  XCOM
        TGT_NODE?  MYNODE
        TGT_USER?  MY_NAME
    TGT_PASSWORD?  MYPSWRD
      TGT_OBJECT?  <CR>
      TGT_DEVICE?  XK0:
       TGT_PHONE?  <CR>
       TGT_SPEED?  9600
        TGT_TYPE?  H
      TGT_SYSTEM?  V
      TGT_PARITY?  N
        TGT_BITS?  8
```

7.3 INVOKING MAIL SERVICES FUNCTIONS

The Mail Services Facility provides several functions that allow a user to handle mail messages. In general, you use the Flow MAIL or EXTERNAL function to access Mail Services functions.

The MAIL function actually translates into a call to FMAIL using the EXTERNAL function. For example, the following function invocations are equivalent:

```
> MAIL CREATE
> EXTERNAL F$MAIL CREATE
```

In the EXTERNAL function, the F$ prefix to the FMAIL taskname indicates that you are calling a task that is accessible only from Flow.

INVOKING MAIL SERVICES FUNCTIONS

The format of the MAIL function is:

MAIL msfunc[/qualifier]

Where:

msfunc is any of the Mail Services functions:

- ANSWER
- BACKGROUND
- CREATE
- FORWARD
- MORE
- READ
- SEND

/qualifier	is any qualifier that is valid for the specified operation.

The following sections describe each of the Mail Services functions.

NOTE

Mail Services functions send and receive only messages that are in the default file cabinet, [ZZDOCO]DEFAULT.DDB. Mail Services cannot access messages that are in any other cabinet.

7.3.1 MAIL ANSWER

Reply to a received message.

Format

MAIL ANSWER

Description

This function allows the user to answer a message that he or she has received. FMAIL performs the following steps:

1. Scan the received message and place the author field into a TO: field in the template. Place the subject from the received message into the SUBJECT: field in the template.

2. Call a Document Services routine to create a message.

3. Call a Document Services routine to place the created message in folder CREATED.

Mail Services uses the author of the received message as the value in the reply's single TO: field. The subject of the reply is the same as the subject of the received message, with "In Reply To" preceding it.

Forms Used by This Function

None.

7.3.2 MAIL BACKGROUND

Enable or disable the wakeup time for TMAIL.

Format

```
MAIL BACKGROUND[/qualifier]
```

Description

Use this function to enable or disable automatic wakeups for TMAIL.

When you establish a TMAIL wakeup time, TMAIL activates at the time you specify, once per day. Upon activating , TMAIL places received mail in folder UNREAD and attempts to send mail that is in folder PENDING_PICKUP.

The qualifiers are:

/DISABLE

Specifies that you want to disable TMAIL from automatically activating. Note that you can manually activate TMAIL simply by executing the function:

```
> ACTIVATE [ZZFLOW]TMAIL.TSK
```

This invocation of ACTIVATE runs TMAIL in the background.

/ENABLE=hh:mm

Where:

hh:mm	is the absolute time that TMAIL will wake up every day. <u>Absolute time</u> is based on a 24-hour clock.

This qualifier specifies that you want to establish a wakeup interval.

Forms Used by This Function

None.

Examples

```
> MAIL BACKGROUND/ENABLE=7:30      ! Wake up at 7:30 AM every day
> MAIL BACKGROUND/ENABLE=22:00     ! Wake up at 10:00 PM every day
> MAIL BACKGROUND/DISABLE          ! Disable automatic wakeup
```

7.3.3 MAIL CREATE

Create a mail message.

Format

MAIL CREATE

Description

This function allows the user to create a mail message. FMAIL performs the following steps:

1. Accepts the following data via an FMS form:

   - TO: - Usernames of people receiving the message.
   - CC: - Usernames of people receiving "copies" of the message.
   - SUBJECT: - The subject of the message.

2. Call a Document Services routine to create the message and edit it. FMAIL places the message in folder CREATED. FMAIL also fills in the TO:, CC:, and SUBJECT: fields of the message.

Forms Used by This Function

FMTO	Allows entry of usernames into the TO: and CC: fields. Displayed using the tag FMAILF.
FMLIST	Menu that displays a list of usernames. Displayed using the tag FMAILM.

NOTE

The symbol FM$TEMPLATE must contain a valid and existing file name, because Mail Services builds the header of the message from this file.

Also, the symbols FM$TO, FM$CC, and FM$SUBJ must be defined because they determine the string that precedes the TO:, CC:, and SUBJ: fields in the message file. Do not change the default definitions of these symbols or mail will not operate properly.

7.3.4 MAIL FORWARD

Forward a mail message to other adressees.

Format

MAIL FORWARD

Description

This function forwards a message that the user has received. FMAIL performs the following steps:

1. Build the list of users to whom you are forwarding the message. This step includes validation of the usernames against the validation file.
2. Call a Document Services function to create and edit the message as a document.
3. Append the message to be forwarded at the end of the list of addresees.

Like the MORE and CREATE functions, FORWARD builds the message header from the standard template file. However, FORWARD appends the received message to this header, forwarding it to users on the new header. The function also allows the user to edit the message to be forwarded.

Forms Used by This Function

FMTO	Allows entry of TO: and CC: information. Displayed using the tag FMAILF.
FMLIST	Menu that displays a list of usernames. Displayed using the tag FMAILM.

NOTE

The symbol FM$TEMPLATE must contain a valid and existing file name, because Mail Services builds the header of the message from this file.

Also, the symbols FM$TO, FM$CC, and FM$SUBJ must be defined because they determine the string that precedes the TO:, CC:, and SUBJ: fields in the message file. Do not change the default definitions of these symbols or mail will not operate properly.

7.3.5 MAIL MORE

Add more TO: and CC: information to an existing message.

Format

MAIL MORE

Description

This function allows the user to add additional TO: and CC: information to an existing message. FMAIL performs the following steps:

1. Obtain more TO: and CC: information from the user via FI.
2. Validate the additional names against the validation list.
3. Parse the message, looking for the last TO:, then add additional TO: fields.
4. Parse the message, looking for the last CC:, then add additional CC: fields.

Forms Used by This Function

FMTO	Allows entry of TO: and CC: information. Displayed using tag FMAILF.
FMLIST	Menu that displays a list of usernames. Displayed using tag FMAILM.

NOTE

The symbol FM$TEMPLATE must contain a valid and existing file name, because Mail Services builds the header of the message from this file.

Also, the symbols FM$TO, FM$CC, and FM$SUBJ must be defined because they determine the string that precedes the TO:, CC:, and SUBJ: fields in the message file. Do not change the default definitions of these symbols or mail will not operate properly.

7.3.6 MAIL READ

Read an unread mail message.

Format

MAIL READ

Description

This function allows the user to read unread mail messages from the folder UNREAD.

FMAIL performs the following steps:

1. Get the first unread message from the folder UNREAD by calling a Document Services routine. Set the message read as the current message by calling another Document Services routine.

2. Place the message in the folder READ.

3. Document Services decrements the value in the symbol OA$MAIL_COUNT, which indicates the number of messages in folder UNREAD. Also, Document Services rebuilds the string OA$MAIL_COUNT_DISPLAY.

4. Display the mail message using the Flow TYPE function.

Forms Used by This Function

None.

7.3.7 MAIL SEND

Place a message in the outgoing queue.

Format

MAIL SEND

Description

This function transfers a message from folder CREATED to folder PENDING_PICKUP, to be sent by FMAIL at its predetermined wakeup time.

You can halt transmission of a message by using the Document Services function DOC MOD to change the folder from PENDING_PICKUP to another folder. You could also use DOC DELETE to place the message in the WASTEBASKET folder. You could even use DOC KILL to remove the message and its header entirely.

There is a very short interval during which TMAIL might have sent the message, but has not yet changed its folder name to SENT. If you attempt to halt transmission during this interval, you will fail.

Note that you can send any type of document that is in DEFAULT.DDB. If there is an error during transmission, TMAIL places the document in folder DEAD_MAIL.

Forms Used by This Function

None.

7.4 MAIL SERVICES SYMBOLS

Table 7-1 describes the symbols used by Mail Services.

Table 7-1: Symbols Used by Mail Services

Symbol	Description
FM$TEMPLATE	This symbol holds the name of the standard template file from which the user creates the message file. Initially it has an equivalence value of "[ZZDOCO]FMTMPLT.DAT".
FM$TO	This symbol contains the default string that appears in the created message file at the beginning of each TO: field. Never redefine this symbol.
FM$CC	This symbol contains the default string that appears in the created message file at the beginning of each CC: field. Never redefine this symbol.
FM$SUBJ	This symbol contains the default string that appears in the created message file at the beginning of each SUBJECT: field. Never redefine this symbol.
OA$MAIL_COUNT	This symbol contains a numeric string indicating the number of currently unread messages (the number of messages in folder UNREAD).
OA$MAIL_COUNT_DISPLAY	This symbol contains a phrase that tells the user the number of currently unread messages.

MAIL SERVICES SYMBOLS

Symbol	Description
TMAIL$_NOTIFY	This symbol determines if and how the user will be alerted that PRO/Office Workstation has received mail. You can define it as follows: ● TERMINAL -- TMAIL sends a message to your terminal screen. ● MESSAGE -- TMAIL sends a message to the P/OS message board. ● BOTH -- Equivalent to both TERMINAL and MESSAGE. ● NONE -- TMAIL does not send a message. By default, this symbol is defined as MESSAGE.
TMAIL$_OBJECT	If defined, this symbol overrides the path name MAIL::*, which represents the path record TMAIL uses for connection to VAX.
TMAIL$_READ	If defined, this symbol overrides the DCL command used on VAX to invoke the ALL-IN-1 read procedure, which is by default: @PROA1:PROA1 "COMMAND PROREAD.COM"
TMAIL$_SEND	If defined, this symbol overrides the DCL command used on VAX to invoke the ALL-IN-1 send procedure, which is by default: @PROA1:PROA1 "COMMAND PROSEND.COM"
TMAIL$_OFF	You can define this symbol to enable or disable the TMAIL task from running in the background. An equivalence value of "Y" disables TMAIL; any other value (or if the symbol is undefined) causes TMAIL to run normally when activated.

Symbol	Description
OA$XKO_RESERVED	The TMAIL task creates this symbol in conjunction with other programs or procedures that use XKO:. If this symbol is non-null, TMAIL is not currently sending or receiving mail. When TMAIL is active, however, this symbol exists with a value of TMAIL date time After TMAIL completes, it redefines the symbol to a null length.
TMAIL$_LOGIN	Setting this symbol to an equivalence of "N" prevents TMAIL from logging you out if you happen to be logged into another account. Instead, TMAIL stops running and leaves you logged in. If the symbol is undefined or has a value other than "Y", then TMAIL does the following: 1. Logs you out of your current account (if this account is not your mail account). 2. Log you into the mail account. 3. Carries out the mail processing procedures. 4. Logs you out of the mail account.

Symbol	Description
TMAIL$_LOGOFF	If TMAIL finds the user logged into the mail account, then it creates this symbol, giving it a value of "N". The module that performs the logout after mail processing then checks for the definition of this symbol. If its equivalence is "N" then it leaves the user in the logged in state (the state that the user was in before TMAIL started up). Note that the automatic background running of TMAIL only occurs when the user is working on the PRO. TMAIL does not start up if the user is in terminal emulation.

CHAPTER 8

USING THE NETWORK SERVICES FACILITY

The Network Services Facility ("Network Services") enables users to connect and login to remote systems through the Professional's Communication Port (device XK0:). The facility provides the following capabilities:

- Background file transfer from Professional to host, host to Professional, or Professional to Professional.
- Distributed processing networks of cooperating personal workstations and host systems.
- Automatic login to remote host system.

Table 8-1 describes some terms we use in this chapter. The remainder of the chapter describes the Network Services database and tasks, the functions that allow you to perform network operations, and the symbols that the facility uses.

Table 8-1: Network Services Terminology

Term	Description
Node	The name of a particular computer system. Example: MOSES.
Object	A process, task, program, or image that you can execute on a remote system. Examples: M_FRIEDMAN (process), MYPROG.TSK (task).
Target	A remote system to which you can connect and login. You identify a target by its node::object combination. Example: MOSES::M_FRIEDMAN.
Account	A user's account on a remote system. Example: MFRIEDMAN.
Path	A record containing information necessary to reach a target. The path name is usually the same as the target name. Example: MOSES::M_FRIEDMAN.
Path Library	A collection of paths stored for subsequent recall. Example: DW1:[ZZOASYM]SYSSYSTEM.SYM contains path records.

8.1 ORGANIZATION OF THE NETWORK SERVICES DATABASE

Network Services maintains a database whose data records consist of paths. A path is a set of information that describes how Network Services can connect and login to a target, and run objects on the target. A path contains such information as:

- Node information (node name).
- User's account information (username, password).
- Circuit Information (circuit device and speed).

Network Services stores path information as symbols in a path library. The path library itself consists of a portion of the system symbol table, [ZZOASYM]SYSSYSTEM.SYM. To access the symbols in the path library, you must treat them as paths by using one of the Network Services tasks, called XLIB. Section 8.2 describes XLIB, as well as the other tasks provided by Network Services.

Each path consists of a symbol whose equivalence value is a 412-byte (maximum length) string. When accessing a path, Network Services maps the equivalence string onto a record definition, as shown below:

```
        map (path)                                           &
                 string path_block=412

        map (path)                                           &
                 string path_dst_nod=20                      &
                ,string path_dst_obj=10                      &
                                                             &
                ,string path_fac_cod=1                       &
                ,string fill=1                               &
                ,string path_fac_nod=20                      &
                ,string path_fac_obj=10                      &
                                                             &
                ,string path_tgt_nod=20                      &
                ,string path_tgt_usr=30                      &
                ,string path_tgt_psw=10                      &
                ,string path_tgt_obj=130                     &
                ,string path_tgt_dev=10                      &
                ,string path_tgt_phn=40                      &
                ,word   path_tgt_spd                         &
                ,string path_tgt_typ=1                       &
                ,string path_tgt_sys=1                       &
                ,string path_tgt_par=1                       &
                ,string path_tgt_bit=1
```

Table 8-2 describes each field in the record definition.

Table 8-2: Description of Path Record Fields

Field	Size	Description
Path Node	20 Bytes	This is a name that defines your path node. The combination of node::object defines the name of a particular path that Network Services uses for recall.

Field	Size	Description
(Continued.)		Usually, you specify your node name on the target VAX in this field (the same as the value in the Target Node field). However, you can specify any name you want. Using XLIB, you specify this value in response to the prompt "Enter Node?".
Path Object	20 Bytes	This is a name that defines your path object. The combination of node::object defines the name of a particular path that Network Services uses for recall. Usually, you specify your username on the target VAX in this field (the same as the value in the Target Object field). However, you can specify any name you want. Using XLIB, you specify this value in the response to the prompt "Enter Object?".
Facility Code	1 Byte	The facility code specifies how Network Services communicates to the target system. There are three codes: 0 -- Reserved. 1 -- Reserved. 2 -- Use the XCOM task to reach remote targets via the Communication Port. Version 1.0 of PRO/Office Workstation supports only code 2, XCOM. This is the actual program that manages the Communication Port on the Professional 350. You must specify 2 for this field in response to the XLIB prompt "FAC_CODE?".
Facility Node	20 Bytes	Reserved.
Facility Object	10 Bytes	The facility object is the name of the facility object (program, task) you are using for communication. In PRO/Office Workstation, this is XCOM. Specify XCOM in response to the XLIB prompt "FAC_OBJECT?".

Field	Size	Description
Target Node	20 Bytes	The target node is the node you want to reach. XCOM ensures that it connects to the proper node based on this field. If XCOM currently has the Communication Port connected to another node, it will disconnect and reconnect to the proper one. If XCOM has the Communication Port connected to the same node, it does not disconnect and reconnect. Note that you usually duplicate the value you specify here in the Path Node field, although that is not necessary. An example of a node is the MOSES portion of MOSES::M_FRIEDMAN. Set this field to your host node name in response to the XLIB prompt "TGT_NODE?".
Target Username	30 Bytes	The target username allows you to login to the target system. This is your username on the host. Note that you usually duplicate the value you specify here in the Path Object field. An example of a username is the M_FRIEDMAN portion of MOSES::M_FRIEDMAN. Specify this value in response to the XLIB prompt "TGT_USER?".
Target Password	10 Bytes	The target password allows you to login to the target system. This is your password on the host. Specify this value in response to the XLIB prompt "TGT_PASSWORD?".

Field	Size	Description
Target Object	130 Bytes	The target object field specifies a DCL command that XCOM invokes on the target system after completing the login. You can enter the value in this field either through XLIB, or via the NETWORK RUN command. For example, in XLIB enter your DCL command in response to the prompt "TGT_OBJECT?". Via the NETWORK RUN command, enter: > NETWORK RUN CRVAX1::WOODS/CMD="DIR" This command connects to the node and logs in to the account specified by CRVAX1::WOODS, and passes the DCL command DIR as the target object (DCL command) to be executed on the VAX.
Target Device	10 Bytes	The target device is the device through which you connect to the target system. For the Professional, this value is XK0:, the device name of the Communication Port. Enter the name in response to the XLIB prompt "TGT_DEVICE?".
Target Phone	40 Bytes	The target phone field provides information that XCOM uses during connection. If your target type is A (autodial modem), specify the phone number of your host VAX. If your target type is G (Gandalf) or I (Micom), specify your node or class number. If your target type is H (hardwired), then leave this field blank by simply pressing the RETURN key in response to the XLIB prompt "TGT_PHONE?".
Target Speed	1-word Integer	The target speed specifies the baud rate at which your host communicates. Enter 9600 in response to the XLIB prompt "TGT_SPEED?".

Field	Size	Description
Target Type	1 Byte	The target type defines the port hardware type. XCOM uses this field to determine what it has to do when connecting and disconnecting the port. Specify one of the following codes in response to the XLIB prompt "TGT_TYPE?": A -- Autodial the attached DF03-AC modem, using the contents of the target phone field. G -- Gandalf switch. Activate the switch, search for "ENTER CLASS", reply with the target phone field, and search for "START". H -- Hardwired. I -- Micom switch. Activate the switch, search for "ENTER CLASS", reply with the target phone field, and search for "GO". U -- User (manual) dial.
Target System	1 Byte	The target system field specifies the operating system of the target computer. XCOM uses this field to determine how to login, logout, run, and finish objects. Specify "V" (for VMS) in response to the XLIB prompt "TGT_SYSTEM?".
Target Parity	1 Byte	Target parity defines the parity used for the connection. Since XCOM always uses <u>no</u> parity, you must set this field to "N" in response to the XLIB prompt "TGT_PARITY?".
Target Bits Per Character	1 Byte	Target bits per character defines the number of data bits in each character on the connection. When making connections to a host VAX, you should set it to "8" in response to the XLIB prompt "TGT_BITS?".

8.2 ORGANIZATION OF NETWORK SERVICES TASKS

The Networks Services Facility consists of two major software components:

- **The XLIB Task**

  XLIB is a task that manages path records in the path library. You invoke this task in order to create or modify a path record. When you invoke XLIB, it prompts you for the contents of each field in a path record, as described in Table 8-2.

- **The XCOM Task**

  XCOM is a communication facility that provides communication via the Professional's Communication Port. XCOM executes routines that are part of Flow to read the information stored in a path record, and uses that information to connect and login to the target system. You invoke XCOM with the Flow function NETWORK.

Figure 8-1 illustrates the organization of the Network Services tasks. The figure shows XLIB and XCOM, which both access the path library contained in [ZZOASYM]SYSSYTEM.SYS. Note that XCOM uses Flow routines to read the path records, while XLIB reads the path records directly.

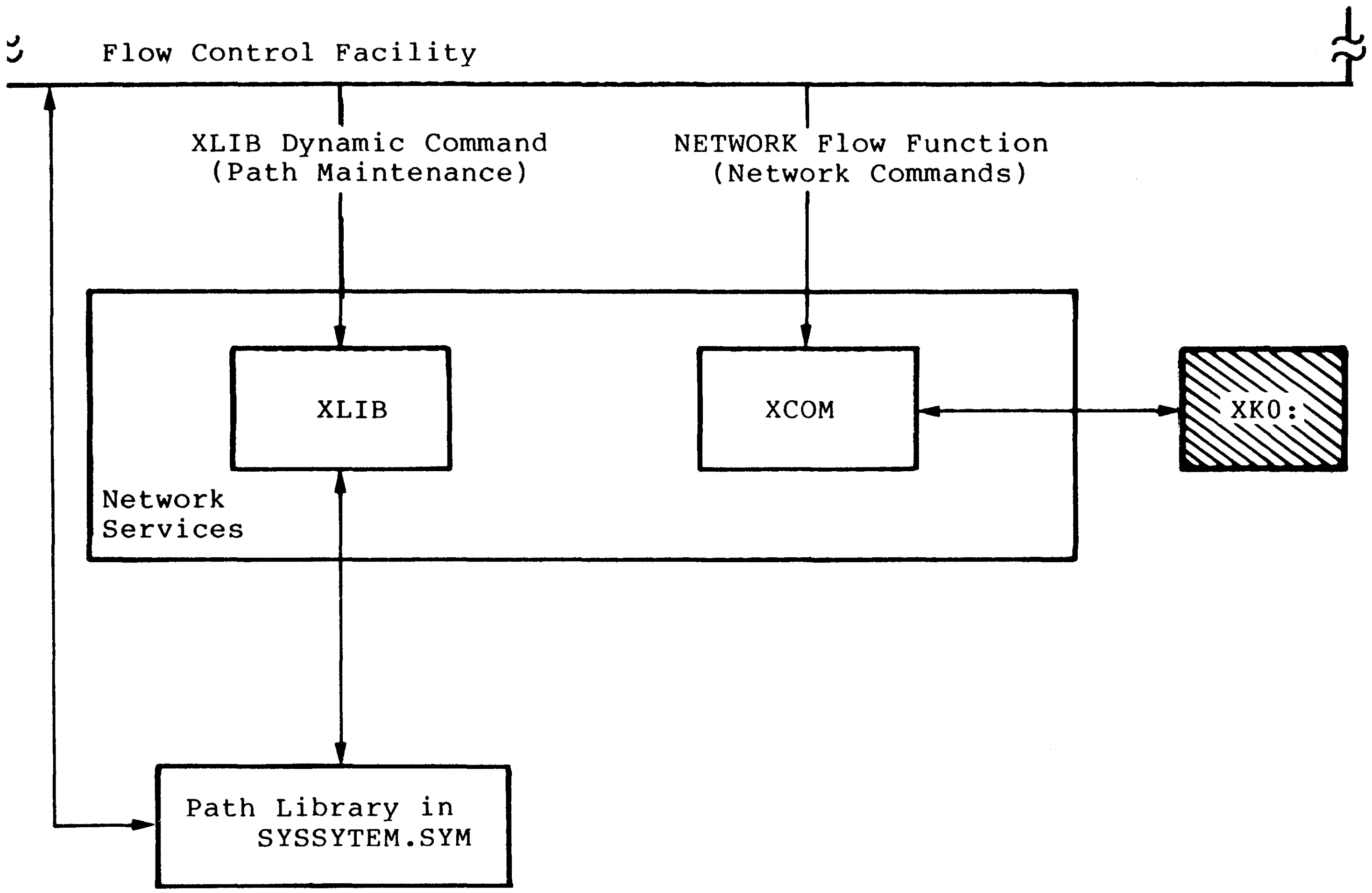


Figure 8-1: Organization of the Network Services Facility

8.3 INVOKING XLIB FUNCTIONS (PATH MAINTENANCE)

XLIB is the task that allows you to create and maintain path records.

To invoke XLIB from CLI, you must first install the XLIB task:

```
> INSTALL [ZZFLOW]XLIB.TSK
```

Then you can create a dynamic command to invoke XLIB:

```
> DEFINE/COMMAND/SYS  XLIB "RUN/TASK DW1:[ZZXNET]XLIB-
>_/NAME=XLIB/COM=""XLIB "
```

Finally, you can invoke the dynamic command XLIB as follows:

```
> XLIB libfunction
```

Where:

libfunction	is a library function that you want XLIB to perform.

The library functions are:

- CREATE (create a path record)
- DELETE (delete a path record)
- DISPLAY (display a specific path record)
- LIST (list all path records)

Assume that you want to create a path library to login to a particular VAX. From CLI, enter the command:

```
> XLIB CREATE
```

XLIB then prompts you to "Select a Library Record" and then to "Create a Path Record." XLIB writes the information you supply into the new path record.

8.3.1 Sample XLIB Session: Hardwired Connection

A sample XLIB session follows. The session assumes that:

- Your facility code is 2 for XCOM.
- Your facility object is XCOM.
- Your target node is MOSES.
- Your target username M_FRIEDMAN.
- Your target password is XYZZY.
- Your target object is the DCL command @LOGIN.
- Your target device is XK0:.
- Your target speed is 9600.
- Your target type is a hardwired connection.
- Your target system a VAX running VMS.
- Your target parity is OFF.
- Your number of bits per character is 8.

```
Select a library record

Enter Node?      MOSES
Enter Object?    M_FRIEDMAN

Create a path record

        FAC_CODE?  2
        FAC_NODE?  <CR>
      FAC_OBJECT?  XCOM
        TGT_NODE?  MOSES
        TGT_USER?  M_FRIEDMAN
    TGT_PASSWORD?  xyzzy
      TGT_OBJECT?  @LOGIN
      TGT_DEVICE?  XK0:
       TGT_PHONE?  <CR>
       TGT_SPEED?  9600
        TGT_TYPE?  H
      TGT_SYSTEM?  V
      TGT_PARITY?  N
        TGT_BITS?  8
```

8.3.2 Sample XLIB Session: Gandalf or Micom Connection

A sample XLIB session follows. The session assumes that:

- Your facility code is 2 for XCOM.
- Your facility object is XCOM.
- Your target node is MOSES.
- Your target username M_FRIEDMAN.
- Your target password is XYZZY.
- Your target object is the DCL command @LOGIN.
- Your target device is XK0:.
- Your target phone is MOSES, the system name that you enter when Gandalf prompts ENTER SYSTEM NAME.
- Your target speed is 9600.
- Your target type is connection via a Gandalf switch.
- Your target system a VAX running VMS.
- Your target parity is OFF.
- Your number of bits per character is 8.

```
Select a library record

Enter Node?     MOSES
Enter Object?   M_FRIEDMAN

Create a path record

        FAC_CODE?  2
        FAC_NODE?  <CR>
      FAC_OBJECT?  XCOM
        TGT_NODE?  MOSES
        TGT_USER?  M_FRIEDMAN
    TGT_PASSWORD?  xyzzy
      TGT_OBJECT?  @LOGIN
      TGT_DEVICE?  XK0:
       TGT_PHONE?  MOSES
       TGT_SPEED?  9600
        TGT_TYPE?  G
      TGT_SYSTEM?  V
      TGT_PARITY?  N
        TGT_BITS?  8
```

NOTE

> PRO/Office Workstation does not provide support for <u>every</u> Gandalf or Micom switch, since most switches are programmed in a unique way. If the procedures we describe here for connecting via Gandalf or Micom do not work for you, contact your Software Services representative for assistance in modifying the XCOM task.

If you are connected to your host with a Gandalf or Micom switch, you must pay particular attention to following items:

- The prompt message that your switch emits to ask you for your class (or system name).
- The actual class (or system name) that you supply to the switch so that it can route you to your host machine.
- The start message that the switch emits once it has connected you to the host.

For example, suppose the switch prompts you for a class when you attempt to connect:

```
ENTER CLASS
```

When connecting you to your host, XCOM waits for this prompt. XCOM compares the last word in the prompt with the equivalence in either of the system symbols XCOM$GANDALF_CLASS or XCOM$MICOM_CLASS. Thus, if the appropriate symbol (depending on your switch) has an equivalence of CLASS, then XCOM can recognize the prompt ENTER CLASS.

It is possible that your switch has been programmed to prompt you for a system (node) name instead of a class number:

```
ENTER SYSTEM NAME
```

If you receive this prompt you must change the equivalence of the appropriate symbol so that it is equal to the last word of the prompt. In this case, invoke DEFINE/SYMBOL as follows:

```
> DEFINE/SYMBOL/SYSTEM XCOM$GANDALF_CLASS "NAME"
```

or

```
> DEFINE/SYMBOL/SYSTEM XCOM$MICOM_CLASS "NAME"
```

After comparing the switch prompt with the symbol equivalence and finding a match, XCOM can supply the switch with the required value. XCOM reads this from the target phone field. Thus, you must enter your normal response to the switch prompt when asked for "TGT_PHONE?" in XLIB. When connecting to the requested machine, XCOM will supply your switch with the value contained in the target phone field.

Two other system symbols, XCOM$GANDALF_START and XCOM$MICOM_START, contain the switch's start message. The start message is a message that your switch emits when it has successfully connected to the requested machine. XCOM reads the appropriate symbol to determine the expected start message. This allows XCOM to know when it is succesfully connected to the target system, so that it can begin the login procedure.

You should set the value of the appropriate symbol to the last word of your start message. For example, suppose your start message is:

```
CLASS 555 START
```

In this case, your symbol should be defined as the follows:

```
> SYMBOL/DEFINE/SYSTEM XCOM$GANDALF_START "START"
```

or

```
> SYMBOL/DEFINE/SYSTEM XCOM$MICOM_START "START"
```

8.4 INVOKING NETWORK SERVICES FUNCTIONS

You invoke Network Services by invoking Flow's NETWORK function and passing a subsystem function as a parameter. The format of the NETWORK function is

```
NETWORK[/qualifier...] nsfunc
```

Where:

nsfunc is one of the Network Services functions:

- CONNECT
- DISCONNECT
- FINISH
- LOGIN
- LOGOUT
- RUN
- START
- STOP

The qualifiers are:

/TIMEOUT=n	The value of n is the NETWORK call timeout. By default, n equals 60 seconds.
/TERMINAL	Run the Dumb Terminal Emulator (DTE) immediately after executing the Network Services function, and exit the Emulator when the NETWORK function is done.

Also, you can specify any of the global qualifiers described at the beginning of Chapter 3.

In order to invoke any of the Network Services functions, you must first invoke NETWORK START.

Table 8-3 shows the how the functions allow you to engage and disengage a target at varying levels. Sections following the table describe all the the Network Services functions.

Table 8-3: Engaging and Disengaging Targets

Engage	Disengage	Description
CONNECT	DISCONNECT	The first level of engagement is connecting to the target. Once connected, you can login to an account on the target. Once logged in, you can run an object on the target.
LOGIN	LOGOUT	The second level of enagement is logging in to the target. You must be connected in order to login. Once you are logged in, you can run an object on the target.
RUN	FINISH	The final level of engagement is running an object on the target. You must be connected and logged in to do this.

8.4.1 NETWORK CONNECT

Connect to a target.

Format

```
NETWORK CONNECT node::object
```

Where:

node::object	is the node::object pair you specified as the name of your path record in the "Select a Library Record" portion of an XLIB session. This is the name of your target.

Description

The CONNECT function performs the first level of engagement to a remote target. Once connected, you can LOGIN to the target and then RUN an object.

Example

```
> NETWORK CONNECT MOSES::MFRIEDMAN
```

8.4.2 NETWORK DISCONNECT

Disconnect from a target.

Format

```
NETWORK DISCONNECT      node::object
```

Where:

node::object	is the node::object pair you specified as the name of your path record in the "Select a Library Record" portion of an XLIB session. This is the name of your target.

Description

The DISCONNECT function disengages from a connected remote target. If you are logged in or running an object, the DISCONNECT also performs a LOGOUT and a FINISH.

Example

```
> NETWORK DISCONNECT MOSES::MFRIEDMAN
```

8.4.3 NETWORK FINISH

Stop a running object.

Format

```
NETWORK FINISH  node::object
```

Where:

node::object	is the node::object pair you specified as the name of your path record in the "Select a Library Record" portion of an XLIB session. This is the name of your target.

Description

The FINISH function only stops a running remote object. It does not LOGOUT or DISCONNECT from the target.

Example

```
> NETWORK FINISH MOSES::MFRIEDMAN
```

8.4.4 NETWORK LOGIN

Login to a target.

Format

```
NETWORK LOGIN   node::object
```

Where:

node::object	is the node::object pair you specified as the name of your path record in the "Select a Library Record" portion of an XLIB session. This is the name of your target.

Description

The LOGIN function performs two levels of engagement to a remote target: it connects and logs in. You can then issue the NETWORK RUN command to run an object.

Example

```
> NETWORK LOGIN MOSES::MFRIEDMAN
```

8.4.5 NETWORK LOGOUT

Logout from a target.

Format

```
NETWORK LOGOUT  node::object
```

Where:

node::object	is the node::object pair you specified as the name of your path record in the "Select a Library Record" portion of an XLIB session. This is the name of your target.

Description

The LOGOUT function stops a running object and logs out from the target. It does not DISCONNECT from the target.

Example

```
> NETWORK LOGOUT MOSES::MFRIEDMAN
```

8.4.6 NETWORK START

Begin XCOM processing.

Format

NETWORK START [dirspec]

Where:

dirspec	is the directory name, enclosed in square brackets. This is the directory on the local system that contains subsystem tasks. By default, this value is [ZZXNET].

Description

The NETWORK START function allows you to install the XCOM task. A warning occurs if the task is already installed. Any other errors cause the function to fail.

Example

Suppose you have already defined a path in a path library, as in the example in the section 8.3.1. You can login to the target system defined in that path by specifying the following commands:

```
> NETWORK START                           !start up XCOM
> NETWORK/TERM LOGIN MOSES::M_FRIEDMAN    !login, run DTE
```

8.4.7 NETWORK STOP

Stops the XCOM task.

Format

NETWORK STOP

Description

The NETWORK STOP command stops the XCOM task. It sends a Network Services request to stop the task (not a P/OS stop). It does not remove the background tasks.

You should not stop the subsystem unless it is actually active; otherwise, multiple stop messages may be queued to XCOM when it is not active. One method to correct this is to issue a NETWORK START before every NETWORK STOP. NETWORK START commands are harmless if the subsystem is already active.

Example

```
> NET START
> NET STOP
```

8.5 REMOTE PRINTING OF DOCUMENTS

PRO/Office Workstation provides two command procedures that allow you to print documents on a remote printer. These procedures are:

- **[VAXCOM]REMPRT.COM**

  You execute this command procedure on the VAX. You must place this file in OALIB: on the VAX, and define the logical PROA1: to point to OALIB:.

- **[ZZFLOW]VAXPRINT.COM**

  Execute this command procedure from Flow to print remotely. It executes the OALIB:REMPRT.COM procedure on the host VAX.

8.6 COMMUNICATING WITH VAX ALL-IN-1

This section describes several command procedures that Network Services and Mail Services use to communicate with VAX ALL-IN-1. Also, the section illustrates how you can login to a VAX ALL-IN-1 system from your Professional.

Note that you must have defined the following path record for Mail Services to communicate with VAX ALL-IN-1:

MAIL::*

For more details, see Chapter 7, Section 7.2.1.1.

8.6.1 VAX Command Procedures

PRO/Office Workstation uses several command procedures to communicate with VAX ALL-IN-1. The procedures execute on VAX under DCL. They are present on the PRO/Office Workstation kit in OA1:[VAXCOM]*.*, but are not copied onto the PRO during installation.

To properly invoke any of the command procedures, you must first define the logical name PROA1: on the host VAX; this logical must point to the OALIB: directory, where you must place all of the command procedures.

On VAX:

```
$ ASSIGN OALIB: PROA1:
```

A description of each file follows:

- **PROA1.COM**

  Starts VAX ALL-IN-1 to communicate with the workstation. Invoked by the NETWORK RUN command from Flow on the PRO.

- **PROCAL.COM**

  Performs the VAX ALL-IN-1 calendar function. Invoked by NETWORK RUN command from Flow on the PRO.

- **PROSEND.COM**

  Sends mail from the PRO to VAX ALL-IN-1. Invoked by the task TMAIL on the PRO.

- **PROREAD.COM**

  Reads mail from VAX ALL-IN-1 and places it on the PRO. Invoked by the task TMAIL on the PRO or PROSEND.COM on VAX.

- **PROGETDOC.COM**

  Selects a VAX ALL-IN-1 document and transfers it to the PRO. Invoked by the procedure VAXGET.COM on the PRO.

- **PROPUTDOC.COM**

  Inserts a document from the PRO database into the VAX ALL-IN-1 database. Invoked by the procedure VAXPUT.COM on the PRO.

- **PROMEXT.COM**

  Creates a Workstation-compatible mail validation file from the VAX ALL-IN-1 database. (Uses PROMEXT.DTR.)

- **PROMEXT.DTR**

  Datatrieve procedure used by PROMEXT.COM.

Please note that the PROMEXT functionality requires that you have Datatrieve installed on the VAX.

8.6.2 Sample Session: Running VAX ALL-IN-1 Through DTE

This section describes how to run ALL-IN-1 on a host VAX via DTE. DTE is the name of the PRO/Communications Dumb Terminal Emulator.

Define the XLIB dynamic command as shown earlier in this chapter. Enter XLIB CREATE from CLI to create a path record. Make sure that you respond to the "TGT_OBJECT?" prompt by typing A1. We are assuming that you would normally type A1 on your VAX to start up ALL-IN-1.

A1 is the program name for ALL-IN-1 on the VAX. XCOM allows you to automatically execute a program (object) on the VAX. In this case, we have chosen ALL-IN-1.

Now, build this function into a command file. You might want to put the file in a directory called [COMMANDS] with other command procedures. Call the file A1.COM and use EDT to create it:

```
$ !
$ ! DW1:[COMMANDS]A1.COM
$ !
$ CLEAR
$ WRITE SYS$OUTPUT "Calling ALL-IN-1..."
$ NETWORK/QUIET START
$ NETWORK/TERMINAL RUN node::object
$ NETWORK DISCONNECT
```

Test it by typing

```
> @[COMMANDS]A1.
```

NOTE

> If your VAX is heavily loaded, you might get a timeout error. At the > prompt, type NETWORK STOP and then try again.

A word of caution: XCOM will cause a login sequence into your VAX and then simulate your typing A1. VAX ALL-IN-1 will begin its start-up processing. At the same time, FLOW will run the terminal emulator (due to the /TERMINAL qualifier). The timing of these two events can cause unpredictable results. This would occur, for example, if VAX ALL-IN-1 performs its startup before DTE performs its startup. However, there is a solution to this problem.

Instead of using "A1" as the TGTOBJECT, use @MYA1. Then put the DCL command file MYA1.COM in your VAX account:

```
$!
$! MYA1.COM in your VAX account
$!
$! This command file will wait 5 seconds before
$! invoking ALL-IN-1 to close the window between
$! DTE and ALL-IN-1 start-up processing.
$!
$WAIT 00:00:05          !Wait 5 seconds
$A1                     !Start up ALL-IN-1
$LO                     !Log-off and get back to FLOW by
$                       !pressing the EXIT key
```

The command procedure forces the VAX to wait 5 seconds before executing the VAX ALL-IN-1 startup processing, thus ensuring that DTE has begun execution.

If you like, you can use the DEFINE/KEY command to load the command @DW1:[COMMANDS]A1 into a function key. For example:

```
> DEFINE/KEY 19 "@DW1:[COMMANDS]A1"
```

Now you can invoke the procedure A1.COM by pressing the F19 function key.

8.7 NETWORK SERVICES SYMBOLS

Table 8-4 describes the symbols that Network Services uses. Note that, in particular, Network Services uses the first three symbols shown in the table to determine the current state of the Communication Port.

Table 8-4: Symbols Used by Network Services

Symbol	Description
XNET$XCOM_NODE	This symbol contains the name of the node to which the user is connected. Network Services defines a non-null equivalence for this symbol only if the user is connected to the target system. Otherwise the symbol is defined as null.

Symbol	Description
XNET$XCOM_USER	This symbol contains the username of the user who is connected and logged in to a target. Network Services defines a non-null equivalence for this symbol only if the user is connected and logged in to the target. Otherwise the symbol is defined as null.
XNET$XCOM_OBJECT	This symbol contains the name of an object that is currently running on a target. Network Services defines a non-null equivalence for this symbol only if the user is connected, logged in, and running an object on the target. Otherwise the symbol is defined as null.
XCOM$GANDALF_CLASS	This symbol must contain the last word of the prompt that your Gandalf switch emits when you are connecting to a target.
XCOM$MICOM_CLASS	This symbol must contain the last word of the prompt that your Micom switch emits when you are connecting to a target.
XCOM$GANDALF_START	This symbol must contain the last word of the start message that your Gandalf switch emits when you are connecting to a target.
XCOM$MICOM_START	This symbol must contain the last word of the start message that your Micom switch emits when you are connecting to a target.

APPENDIX A

ERROR MESSAGES

A.1 ERRORS GENERATED BY DSI

INSUFDOCIN	You must enter a title and a folder
INSUFCABIN	Please enter a title and a filename
SELSIZERR	Can't fit all choices on screen
SELNOTHING	You have nothing to select
PREVMENUERR	Cannot display previous menu
MAXCACHE	Too many options to display, Truncating
BADCABDEL	Unable to delete cabinet from System cabinet
CABDELFAIL	Unable to delete current cabinet
CABLOCFAIL	Unable to locate cabinet
DELNOCAB	You have no current cabinet to kill
NOSYSCAB	You have no system cabinet
KILNOFIL	You have no current document to kill
NOCURCAB	You have no current cabinet to access
NOSYSCAB	Unable to locate the system cabinet
DOCKILFAIL	Error is attempting to kill document
MODNODOC	You have no current document to modify
DELNODOC	You have no current document to delete
DISNODOC	You have no current document to display
PRTNOFIL	You have no current document to print
FI_FAIL	Unable to process form, exiting
BAD_CREATE	Unable to create document
KILFAIL	Unable to kill document
ILLCABREQ	Unknown cabinet function specified
NOPARM	Insufficient information for create
BADCABCRE	Unable to create cabinet
MODFAIL	Unable to change document
NOFOLD	Unable to locate folder
EDNOFIL	You have no current document to edit
NOUPDATE	Unable to modify document
BADDOC	Unable to retrieve document
CABOPN	Unable to access cabinet
CABGETFAIL	Unable to retrieve cabinet
BADCABSEL	Unable to complete cabinet selection

ERRORS GENERATED BY DSI

DOCGETFAIL	Unable to retrieve document
SELSYM	Unable to select document
SELFAIL	Selection failed
CDSFAIL	Document server has had an error
USRXIT	Exiting by your request
SELILLREQ	Selection criterion not allowed
BADPBUNL	Can't unload a parameter from parameter block
BADCMDLIN	Flow couldn't start the editor
BADPBADD	Unable to address Parameter Block
SYMFAIL	Symbol server has had an error
BADPBLOAD	Unable to load a parameter into the Parameter Block
NOCDSST	Unable to retrieve status from the document server
UNKNOWN_COMMAND	Document Services entered by unknown command
DIRECTIVE_ERROR	Directive error spooling document
SPAWN_ERROR	Unable to start (spawn) editor
PRINTER_MISC	Error accessing printer:
PRINTER_UNKNOWN	Unknown status code from print spool
PRINT_0	Print request failure
PRINT_11	Print job already active
PRINT_10	Printer busy
PRINT_9	Printer already attached
PRINT_8	No print job is active
PRINT_7	Print job is not paused
PRINT_6	Print job already paused
PRINT_5	Parameter out of range
PRINT_4	Printer not connected
PRINT_3	Undefined
PRINT_2	Printer not connected and paused
PRINT_1	Unable to connect to service task

A.2 ERRORS GENERATED BY CDS

BADSYM	Unable to retrieve symbol
SYMNOUPD	Unable to update the symbol
BP2ERR	Basic-Plus-Two error data
BP2TXT	Basic-Plus-Two error text:
CABOPN	Error opening cabinet
CABCRE	Error creating cabinet
BADCAB	Invalid cabinet
BADPRI	Invalid primary document header
BADSEC	Invalid secondary document header
GETHDR	Could not get document header
GDNERR	Could not get next document number in cabinet

ERRORS GENERATED BY CDS

ERRCRE	Create operation failed
ERRDEL	Delete operation failed
ERRGET	Retrieve operation failed
ERRMOD	Modify operation failed
NOCAB	No cabinet argument in call
NOREQ	No request argument in call
ILLARG	Invalid argument in call
ILLREQ	Invalid request
ILLNBR	Numeric value is not numeric

A.3 ERRORS GENERATED BY BASIC-PLUS-2

BP2ERR	Basic-Plus-Two error occurred:
BP2TXT	Text of BP2 error:

A.4 ERRORS GENERATED BY LOGICAL HANDLING

LOG$NAME	Logical name:
LOG$VALUE	Logical equivalence:
SETDEF$SPEC	Default directory:
LOGERR	Error in prolog operation:

A.5 ERRORS GENERATED BY TMAIL

NEWMAIL New mail has been received.

When TMAIL has been activated and it finds new mail it displays this message.

PROERR A mail error occured that has stopped Mail.

TMAIL starts up in the background and experiences problems that cause it to discontinue. Possible reasons might be

- Unable to log into VAX
- Getting timeouts
- Unable to run ALL-IN-1

ERRORS GENERATED BY TMAIL

SNDERR	A mail error has occured while sending mail. TMAIL attempts to send a message from the PRO to the addressees via VAX ALL-IN-1. If a user does not have an ALL-IN-1 profile on the VAX file then a send error will occur. Also might be caused by a message with an illegal format.
FATAL	Fatal TMAIL error:
BUSERR	Software bus error: A call to the software bus failed.
NOCDSSTAT	No status returned by CDS: TMAIL calls Callable Document Services and no status is returned to TMAIL. CDS should also generate some errors as well.
CDSERR	An error was returned by CDS: TMAIL calls Callable Document Services and and an invalid status is returned to TMAIL. CDS should generate some errors as well.
GETERR	Error while reading file: Error while reading the status files created on the PRO containing info on the messages marked for mailing. Possibly the file has a bad format and TMAIL attempts to read past the EOF marker.
MODFOL	Error modifying message folder: TMAIL calls CDS to modify the folder name and an error occurs.

ERRORS GENERATED BY TMAIL

OPNERR	Error trying to open file: An error is encountered when attempting to open an RMS file. Check that the file has been created in the correct manner (it may have been created in a format not compatible with the most recent OPEN statement).
PUTERR	Error writing to file: Attempting to write a record to a file. The record may be too long.
CABERR	Error opening document cabinet TMAIL calls DS to access mail messages and the cabinet specified does not exist. Check that the cabinet specified exists in [ZZDOC0]*.
RUNERR	Error trying to run ALL-IN-1 on VAX: Network Services finds the DCL prompt on the VAX and attempts to run VAX ALL-IN-1. The error may occur because the user is logged in to the wrong account.
PATHERR	Error finding Network Services path: The path specified in the XCOM call (mail::*) is undefined. The user should define this path record.
LGIERR	Error logging into VAX: The XCOM call to log in to the VAX failed, possibly due to a timeout error. For example, if the VAX is exceptionally slow XCOM may time out because it only waits for a certain number of seconds to receive the 'username' prompt from VAX.

Other TMAIL Errors:

RCVERR	Mail error processing VAX data
NEWSTART	Bad new message START:

ERRORS GENERATED BY TMAIL

NEWWARN	Warning for new message:
NEWERR	Error for new message:
NEWUNREC	Unrecognized new.br.i-16;line:
NEWADDERR	Error creating document for new message:
NOSTAT	VAX Status file could not be found:
STATERR	Error in VAX status data
SNDSTART	Bad sent message START:
SNDWARN	Warning for sent message:
SNDERR	Error for sent message:
SNDUNREC	Unrecognized line:

A.6 ERRORS GENERATED BY FLOW NETWORK COMMAND

PATHPROMPT	Please enter the NETWORK path:
PATHERR	The specified path is not defined
UNSPCOM	Not a valid NETWORK command
NOCOMMAND	No NETWORK command specified
NONODE	No required node name found
ILLNUM	Illegal number in switch
REQCOM	XCOM not installed
COMACT	XCOM already active
NOTOPER	XCOM not running
ERROR	NETWORK error:
FACILITY	NETWORK Facility:
WORD1	Word 1 value:
WORD2	Word 2 value:
WORD3	Word 3 value:
BADSTATE	FLOW state unrecognized
INSTASK	PROTSK error installing task:
TSKERR	Error values:
PREINST	Task is already installed:

A.7 ERRORS GENERATED BY FI

INVAL_REQ	Error - request is invalid This means that the user entered an invalid request. The only valid FI requests are MENU, FORM, and FIELD. The error code for this error is 16640. Retry the operation using a correct request

ERRORS GENERATED BY FI

INVAL_DEF	Error - definition is invalid The user has entered an invalid value for /DEF=. Retry the operation using the correct definition for the FI operation.
INVAL_DIS	Error - display is invalid The user has entered an invalid value for /DIS=. Retry the operation using the correct display for the FI operation.
NODIS	Error - display not specified The user has not specify /DIS= when it is required. Check the tag that you are using for the /DIS entry.
NO_DEF	Error - function not specified The user has not specify /DEF= when it is required. Check the tag that you are using for the /DEF entry.
ARGERR	Error calling '.ARG' routine Probably an FMS error.
FILE_ERR	Error in file operation
PB_EMPTY	Error - parameter block is empty This error occurs if FI calls a dynamic form from an application passing an empty parameter block.
LIBNOTOPEN	Error - library not open FI attempts to display a form without having a form library open.

ERRORS GENERATED BY FI

NO_NDATA	No named data entries for specified form FI attempts to display an AI (/DEF=A1) type form whose named data is empty. Use FED to change the FMS form; add the correct entries to named data.
IMPURE_AREA_ERR	Error in named data area of specified form There is a problem reading the named data of the current form. Check the named data in the form using FED.

Other FI Errors

NOFORM	No Form Name Specified - Exiting
INVALINPUT	Invalid Input - Please Reenter
NOMENU	.MENU area missing
NOFILE	File Must be Specified in Argument - Exiting
FORMERR	Form Name Error - Default to 'FORM1'
TOOMNYOPT	Too Many Options - Ignored
TOOLONG	Output Field Too Long - Truncated
NOFIELD	Field Does not Exist on current form
TOOMNYTXT	Too many Text Messages - Ignored
ALLOWNM	No Match Found - Allow
NOMATCH	No match found for choice - Please reenter
INVALLINE	Invalid record found - Ignored

A.8 ERRORS GENERATED BY FLOW CLI

ILLCMD	Illegal or unrecognized command
ERRINPROC	Error forces procedure exit
INPOPNERR	Error opening sys$input,
SYS$INPUT	Sys$input=
REOPERR	Error reopening sys$input,
CMDINPERR	Error inputing command
ILLCMD	Command syntax error,
DFNERR	Error defining symbol
ILLOP	Illegal operation in expression
ILLIOP	Illegal internal operation value
ILLNBR	Variable is not a valid integer
UNDSYM	Symbol in expression is not defined

A.9 ERRORS GENERATED BY FLOW

CMD$CMD	<Flow version>
CPROMPT	<CLI prompt>

ERRORS GENERATED BY FLOW

FIRST_FUNCTION	menu oa$main
STARTUP_FUNCTION	cont
BASELEVEL	<current baselevel>
VER_WRK	<Worstation version message>
VER_FLOW	<Flow version message>
PAUSE	Press RESUME or any function key to continue
PASSWD	Exit from password form - terminating...
ILLCMD	Illegal or unrecognized command
SUBNOTDEF	Undefined symbol for command substitution
FOPINP	FOP Input error
FOPOPN	FOP Open error
FOPOUT	FOP Output error
FOPFDT	FOP Pro input error
EOF	End of file detected
BADREQ	FLOW request not recognized,
BADSRV	FLOW service not recognized,
BADSTATE	FLOW State not recognized,
NOTIMP	Service not implemented
NOSTACK	FCB stack limit reached, use EXIT or UNWIND
INIERR	Flow initialization error
BP2ERR	Basic-Plus-Two error,
FTLBP2ERR	Fatal Basic-Plus-Two error,
CLEAR	+155.+H+155.+J

A.10 ERRORS GENERATED BY FLOW ROUTINES THAT CALL FI

MNU$TAGNAM	Please enter the form specification:
STAERR	Form interface request failed,
MNUERR	Form interface operation failed

A.11 ERRORS GENERATED BY FMAIL

ERR_CALLING_FI	Error in call to Forms Interface The module that allows you to enter data in TO: and CC: fields has sent back an error status to the caller. Other error messages accompanying this message should explain the exact problem.

ERRORS GENERATED BY FMAIL

ERR_CALLING_FMTO	Error in call to module FMTO The calling routine receives an invalid status back from the "entering TO: and CC:" module. Accompanying error messages should indicate the cause of error in that module.
ERR_CALLING_DS	Error in call to Document Services FMAIL makes a call to Document Services, which returns an invalid status. Accompanying error messages should indicate the cause of the error in Document Services.
ERR_CALLING_FMTMP	Error in call to module FMTMP The calling program receives an invalid status back from the message-building routine. Accompanying error messages should indicate the cause of error in that module.
ERR_CALLING_DSCRE	Error in call to Document Services. FMAIL receives an invalid status from Document Services when attempting to do a DOC CREATE. Accompanying error messages should indicate the cause of error in that module.
ERR_CALLING_DSEDIT	Error in call to Document Services FMAIL receives an invalid status from Document Services when attempting to do a DOC EDIT. Accompanying error messages should indicate the cause of error in that module.
ERR_CALLING_DOCMOD	Error in call to Document Services FMAIL receives an invalid status from Document Services when attempting to do a DOC MODIFY. Accompanying error messages should indicate the cause of error in that module.

BP2_ERR	Run time error occurred A BASIC-PLUS-2 error occurred. The error code received is the BASIC run-time error. This error might occur by using invalid files or files with an illegal format. Look up the error code in the basic run time errors
TEMPLATE_SYM_UNDEF	Template Symbol is undefined The symbol FM$TEMPLATE, which holds the name of the message template file, does not exist in the symbol table. Fix this by creating the symbol and giving it the equivalence of the valid template file.
SWB_ERR	Software Bus error occurred A call on the software bus returns an invalid status. Check parameters and retry.
FILERR	User not in Validation file -- bp2 error Might be the result of a corrupt validation file. Compare the addressees on the validation file to the addresses in the current message.
SUBJ_SYMERR	Subject symbol error The symbol FM$SUBJECT, which holds the field title SUBJECT:, is undefined. Create this symbol.
TO_SYMERR	To symbol error The symbol FM$TO, which holds the field title TO:, is undefined. Create this symbol.
CC_SYMERR	CC symbol error The symbol FM$CC, which holds the field title CC:, is undefined. Create this symbol.

ERRORS GENERATED BY FMAIL

ERR_TRN_SYM	Symbol translation error Symbol Services returns an invalid status. Check for the existence of the symbols it may be translating and create the necessary symbols.
MSG_ILL_FORMAT	Mail message has illegal format When attempting to add more TO: and CC: information to a mail message, the message is found to have an illegal format (possibly it does not have the correct subject -- check the format of the message).
NULL_KEY	Error accessing file -- null key found The author entry in the header for the message is null. Therefore the message is invalid and cannot be operated on. Modify the document's header information.
NO_SUCH_USER	User does not exist on file The addressee name or partial name does not exist on the user validation file on your PRO. If you really want to send mail to this addressee, then you must create an entry for him in the user validation file.
FM$_BADCAB	Mail must be created in cabinet "DEFAULT" Mail messages can only reside in the default cabinet. Therefore, if the current cabinet is anything other than DEFAULT this error occurs. Can be avoided by changing your default cabinet to DEFAULT.
FM$_NOCAB	No current Cabinet This error occurs when you attempt to invoke a mail function without a current cabinet set the current cabinet to DEFAULT.

A.12 ERRORS GENERATED BY PIP

NOINPSP	No input file specification
NOOUTSP	No output file specification
NOTCOMMAND	Unrecognized command

A.13 ERRORS GENERATED BY TYPE

PROMPT	Please Press NEXT SCREEN, PREV SCREEN, MAIN SCREEN, EXIT, PF1, or HELP
HELP1	EXPLANATION OF FUNCTIONS
HELP2	NEXT SCREEN - DISPLAYS NEXT PAGE OF FILE
HELP3	PREV SCREEN - DISPLAYS PREVIOUS PAGE OF FILE
HELP4	MAIN SCREEN - DISPLAYS FIRST PAGE OF FILE
HELP5	EXIT - ENDS TYPE PROGRAM
HELP6	PF1 - 80/132 COLUMN TOGGLE
HELP7	PRESS RETURN TO RESUME TYPING FILE
EOD	[End of Document]
INVAL	INVALID FUNCTION
ERROR1	can't find
ERROR2	Error
ERROR3	(
ERROR4	) at line
ERROR5	On file
FILE	File:
GETERR	in GETMCR call
EXIT	Exiting...

A.14 ERRORS GENERATED BY FLOW CALL OR EXTERNAL COMMANDS

NONZEROESB	Service/module terminated early
SWBERR	Software bus error ocurred,
NOMOREMDB	FLOW external mdb stack exhausted
NOSTAT	Service/module did not return status

A.15 ERRORS GENERATED BY APPLICATION-RELATED COMMANDS

NAME	Please enter FLOW application name
NOSPEC	No application name has been given
USREXI	User chose to exit operation
PA1APL	PA1APL server incurred an error:
IMGEXC	Install file has too many tasks or libraries
SPWNERR	Error spawning task specified in RUN command,
RUNDWN	Please wait while the application is terminated...

CTRLC	Application terminated by user Control-C
UNDEF	This application is not defined in FLOW,
INSOPNERR	Error opening application install file,
DEBOPNERR	Error opening debug output file,
VERCRELOG	creating logical name
VERINSTSK	installing task or library
VERDELLOG	deleting logical name
VERREMTSK	removing task or library
VERABOTSK	aborting task
CRELOG	Error creating logical name
INSTSK	Error installing task or library
DELLOG	Error deleting logical name
REMTSK	Error removing task or library
ABOTSK	Error aborting task
INSINPERR	Error reading install file,
ILLREQ	Illegal request made to f$appl,

A.16 ERRORS GENERATED BY SYMBOL-RELATED COMMANDS

ILLCOM	Illegal operation or symbol type
CREERR	Error creating definition
DELERR	Error deleting definition
GETERR	Error translating definition
DEFSYM$NAM	Symbol name:
DEFSYM$EQV	Symbol equivalence:
DEFCOM$NAM	Dynamic command name:
DEFCOM$EQV	Dynamic command equivalence:
DEFKEY$NAM	Function key number:
DEFKEY$EQV	Function key equivalence:
DEFTAG$NAM	Tag name:
DEFTAG$EQV	Tag equivalence:
DEFAPP$NAM	Application name:
DEFAPP$EQV	P/OS application title:

A.17 ERRORS GENERATED BY TASK-RELATED COMMANDS

NONAME	Task name not specified
SPWTSK	Error running (spawning) task
TSKERR	Spawned task returned error
BASTATE	FLOW state not recognized
NOFILE	No file specification for install
PREINS	Task already installed
NOTINS	Task not installed
PROTASK	PROTSK error
DIRERR	P/OS Directive error

A.18 ERRORS GENERATED BY FLOW VOLUME-RELATED COMMANDS

DIRNAM	Directory specification:
VOLNAM	Volume device label (ddu:label):
VOLMOU	Volume mounted:
VOLDIS	Volume dismounted:
MOUNT	Error mounting volume:
DISMOU	Error dismounting volume:
DIRCRE	Error creating directory:
DIRDEL	Error deleting directory:
USREXI	User aborted operation

APPENDIX B

LIST OF DEFAULT FORMS

Following is a list of all the forms provided in [ZZFLOW]OAFMS.FLB. You can display any of these forms by executing the FORM or MENU function, whichever is appropriate (this depends on whether the form is an argument-type form or a menu-type form).

ACCTSU
APPL
APPLSU
BA
CMS
COMMSU
DEFLIS
DEFMEN
DM
DOCSEL
DSCCB
DSCDC
DSCOP
DSDCB
DSDCBW
DSDDC
DSDDCW
DSDFL
DSDFLW
DSEDCW
DSMDC
DSSCB
DSSDF
DSSFL
DSSNU
DSSTI
EM
EMSU
FCM
FILE

FLOW
FLOWSU
FMLIST
FMTO
FMUSER
FORM
MAIN
MENU
MSGSEL
PASSWD
POS
PROFIL
SUCOM
SUEDIT
SUFIR
SUKEY
SUNOT
SUSYM
SUTAG
SYSSU
USERSU
WP
WSSU
XNET
XNETSU

INDEX

PRO/Office Workstation
Programmer's Manual
Order No. AA-BM02A-TK

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______
Organization ______
Street ______
City ______ State ______ Zip Code ______
or
Country

Please cut along this line.

-- Do Not Tear - Fold Here and Tape --------------------------------

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional 300 Series Publications
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

--- Do Not Tear - Fold Here --------------------------------

Cut Along Dotted Line

0029958

Printed in U.S.A.