PERFORM Reference Guide
PDR 3906

1 INTRODUCTION

1.1 How to Use This Manual

Each pair of facing pages presents a single topic, with a narrative description on the left page and usually some examples on the right page.

Each topic is presented on a pair of facing pages. The left-hand page contains the main text and the right-hand page presents the figures supporting the text. Each left-hand page also has a title followed by a brief summary of the topic presented.

This format has the advantage that a topic is covered in one pair of facing pages and can be learned or referenced quickly.

A set of general conventions has been established to support this format. See Figure 1.

Table of Contents

CONVENTION	MEANING
UPPER CASE	Upper case words or characters are required and must appear exactly as shown.
lower case	Lower case words or characters are parameters to be supplied by the user (i.e., file.name, field.name, etc.)
{}	Words or parameters surrounded by braces indicate that the word or parameter is optional and may be included or omitted at the user's option.
{}...	An ellipsis (i.e., three dots) that follows the terminating bracket means the enclosed word or parameter may be omitted or repeated an arbitrary number of times.
<u>COMMAND</u>	In the examples, words that are underlined are commands and parameters typed at the display terminal by the user.

FIGURE 1. Conventions Used.

1.2 Communicating with the PRIME INFORMATION System

Communication with the PRIME INFORMATION System is accomplished through the use of commands entered at the terminal keyboard in response to a colon prompt (:).

Communication with the PRIME INFORMATION System is accomplished through the use of commands entered at the terminal keyboard in response to PERFORM's prompting. These commands are analyzed and processed by PERFORM as the first step in accomplishing the requested action.

The PERFORM prompt character is a colon (:). When PERFORM prompts for the next request with this colon prompt, commands may be entered via the terminal keyboard. The command may be the name of a VERB, or it may be the name of a stored SENTENCE, PARAGRAPH, or MENU.

A VERB is a command that is defined in the vocabulary file and contains "V" in field one. Verbs invoke various processors within the PRIME INFORMATION System. (See the section titled "PRIME INFORMATION COMMANDS" in this manual.)

A SENTENCE is a complete command to PERFORM which consists of one or more words, the first one of which must be a verb. The simplest complete sentence is simply a verb name. (See the section titled "SENTENCE Definition Records" in this manual.)

A stored SENTENCE is a named item in the VOC file, which consists of a complete sentence formed according to the rules for sentences.

A PARAGRAPH is a named item in the VOC file, which consists of one or more sentences. (See the section titled "PARAGRAPH Definition Records" in this manual.)

A MENU is a named item in the VOC file, which consists of a displayed list of functions which may be selected and, by their selection, executed. (See the section titled "MENU Definition Records" in this manual.)

If the command is typed in upper-case characters, it must be found in the VOC file exactly as typed. If the command is entered in lower-case characters, it will be searched for in the VOC file exactly as typed, and if found, will be executed. However, if a lower-case command is not found, it will be transformed into its upper-case equivalent, and then searched for again in the VOC file; if found in this form, it will be executed.

:<u>LIST ORDERS WITH SHIP.DATE > 11/30/78</u>	A sentence beginning with a verb
:<u>INVENTORY.MENU</u>	The name of a menu
:<u>REPORT.3</u>	The name of a stored sentence or paragraph

FIGURE 1. Examples of Commands.

2 PERFORM COMMAND LANGUAGE

2.1 PERFORM Introduction and Overview

PERFORM accepts words presented to it and, using the VOC file to find the meaning of each word, transforms them into sentences. These sentences then invoke various processors within the PRIME INFORMATION System.

The function of PERFORM is to accept commands from the terminal keyboard, analyze the intended action from the words contained in the commands (using the VOC file as a dictionary for this analysis), and use the resolved command to invoke one of the PRIME INFORMATION System processors.

PERFORM operates on verbs, stored sentences, menus and paragraphs. Its operation is dictated by the type of the first word in the command (after PERFORM has "looked up" the command in the VOC file, to determine its type from data stored there).

If the first word of the command is a verb, PERFORM invokes the proper processor and passes the remainder of the command sentence to that processor.

If the first word of the command is the name of a stored sentence, PERFORM finds the actual stored sentence in the VOC file and uses it to replace the first word of the command (the name of the stored sentence). The command in its new form is then re-analyzed, as if the sentence itself had been entered in response to the colon prompt.

If the first word of the command is the name of a paragraph, the paragraph is found in the VOC file, and each sentence within the paragraph then becomes a sentence to be analyzed by PERFORM, just as if a series of sentences had been entered from the terminal keyboard at successive colon prompts. When the sentences in the paragraph are exhausted, PERFORM issues another colon prompt and waits for the next command to be entered.

If the first word of the command is the name of a menu pointer, the menu named in the VOC file pointer is invoked. As each successive selection is made from the menu, the corresponding command is analysed just as if it had been entered from the keyboard at the colon prompt. After the selected command's function has been completed, control returns to the menu to allow another selection. When no further selections are made from that menu, control is returned to PERFORM, which issues another colon prompt and waits for the next command to be entered at the terminal.

If the first word is:	Action is:
The name of a Verb	Invoke the proper processor and pass the sentence to that processor.
The name of a stored sentence	Extract the named sentence from the vocabulary file and analyze it as if it had been typed by the operator, instead of just its name.
The name of a paragraph	Get the stack of commands from the paragraph and start executing them sequentially.
The name of a menu	Display the menu and prompt for the user to select one of the options.
Anything else	Display "'Anything' is not in your vocabulary file"

FIGURE 1. Summary of PERFORM responses to command types.

2.2 PRIME INFORMATION System Processors

Actions within the PRIME INFORMATION System are performed by various processors, which belong to one of five categories: INFORM processors, DIRECT processors, PERFORM processors, INFO/BASIC processors, and PRIMOS command processors.

Processors which are invoked by PERFORM in response to PERFORM's analysis of the commands entered at the terminal belong to five categories: INFORM processors, DIRECT processors, PERFORM processors, INFO/BASIC processors, and PRIMOS command processors.

INFORM processors are those which access PRIME INFORMATION files via the database manager, using file dictionaries and the data files described by those dictionaries. In general, INFORM processors are those which accomplish the query and report generation functions of PRIME INFORMATION systems. INFORM processors are identifiable in the VOC file by having IN in field three.

DIRECT processors access PRIME INFORMATION files directly, without using the file descriptions found in the file dictionaries. Also included in this category of processors are those which access no files, such as TIME. DIRECT processors are identifiable in the VOC file by having DI in field three.

PERFORM processors are those which interact directly with PERFORM, and must be uniquely identifiable by PERFORM. Examples of this type of processor would be DEBUG, or WHO. PERFORM processors are identifiable in the VOC file by having PE in field three.

INFO/BASIC processors are programs which are written in INFO/BASIC, usually by the user of the PRIME INFORMATION system (although there are some INFO/BASIC processors which are part of PRIME INFORMATION software). INFO/BASIC processors are identifiable in the VOC file by having IB in field three.

PRIMOS processors are commands which are available at the PRIMOS level that have been "brought up" into PRIME INFORMATION. These processors are identifiable in the VOC file by having PR in field three.

These five types of processors (defined by IN, DI, PE, IB or PR in field three of the VOC record of each verb) are the only types defined to PERFORM. Of these five types, only type IB (for INFO/BASIC programs) is available for use by customers. Types IN, DI, PR, and PE are explicitly reserved for use by PRIME INFORMATION software, as are all undefined types.

INFORM processors

COUNT	ENTER	ENTRO	ENTROC
LIST	SELECT	SORT	SSELECT
SUM	T.BCK	T.DUMP	T.FWD
T.LOAD	T.READ	T.REW	T.WEOF

DIRECT processors

ASSIGN	BASIC	BLOCK.PRINT	BLOCK.TERM
BTD	CATALOG	CD	CHAP
CLEAN.ACCOUNT	CLEAR.FILE	CLEAR.LOCKS	CLR
CNAME	COMO	COMPILE.DICT	COPY
CREATE.FILE	CS	DELAY	DELETE
DELETE.CATALOG	DELETE.FILE	DELETE.LIST	DTB
DTO	DTX	ED	FILE.STAT
FORM.LIST	FORMAT	GET.LIST	GROUP.STAT
GROUP.STAT.DETAIL	HASH.HELP	HASH.HELP.DETAIL	HASH.TEST
HASH.TEST.DETAIL	HELP	HUSH	ISTAT
LIST.LOCKS	LIST.READU	LO	LOCK
LOGOUT	MAKE.MAP.FILE	MAP	MASTER
MESSAGE	OFF	OTD	P.ATT
P.DET	PASSWD	PHANTOM	PTIME
RADIX	RECORD	RELEASE	RESET.PRINTER
RESIZE	RUN	SAVE.LIST	SETFILE
SETPTR	SLEEP	SPOOL	STAT
STATUS	T.ATT	T.DET	TERM
TIME	UNASSIGN	USERS	VCATALOG
XTD			

PERFORM processors

CLEARDATA	CLEARSELECT	DEBUG	DISPLAY
GO	IAM	IF	LOGTO
LOOP	QUIT	REPEAT	WHO

INFO/BASIC processors

DATE	DATE.FORMAT	MENU.DOC	MENU.PIX
RELEASE.ALL	RELEASE.ITEMS		

PRIMOS processors

AVAIL	MAIL	PROP	PSPOOL
PTERM			

FIGURE 1. Examples of Various Types of Processors.

2.3 INFORM System Processors

Some of the verbs defined in a PRIME INFORMATION account's VOC file are processors which belong to INFORM. Those verbs are documented in THE INFORM REFERENCE GUIDE, IDR39Ø5.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR39Ø5. INFORM processors which belong to this category are:

COUNT	ENTER	ENTRO
ENTROC	LIST	SELECT
SORT	SSELECT	SUM
T.BCK *	T.DUMP *	T.FWD *
T.LOAD *	T.READ *	T.REW *
T.WEOF *		

INFORM processors are those which access PRIME INFORMATION files via the database manager, using file dictionaries and the data files described by those dictionaries. In general, INFORM processors are those which accomplish the query and report generation functions of PRIME INFORMATION systems. INFORM processors are identifiable in the VOC file by having IN in field three.

Note that several magnetic tape commands are implemented as INFORM processors (those marked with an asterisk above). The T.DUMP and T.LOAD processors utilize the full power of INFORM, and thus are implemented as INFORM processors. The other magnetic tape commands are documented in this manual, but are actually implemented as part of the INFORM system.

INFORM processors

COUNT	ENTER	ENTRO
ENTROC	LIST	SELECT
SORT	SSELECT	SUM
T.BCK *	T.DUMP	T.FWD *
T.LOAD	T.READ *	T.REW *
T.WEOF *		

Commands marked with an asterisk (*) are documented in this manual, even though they are implemented as INFORM processors.

FIGURE 1. Verbs documented in THE INFORM REFERENCE GUIDE, IDR3905.

2.4 Command Invocation Modes

Most PRIME INFORMATION commands have two primary modes of invocation: single-line, where all required parameters are entered on the command line; and prompting, where each omitted parameter is requested by the command processor.

Most commands that are supplied with the PRIME INFORMATION system are responsive to two primary modes of invocation. The first mode is single-line mode, wherein the command and all of its required parameters are typed on a single line. The other mode ("prompting" mode) requires only that the command itself be typed; omitted parameters are requested via prompts from the command processor.

For an experienced user, the single-line mode will probably be preferred. Since many of the parameters are position-dependent (rather than being keywords), only an experienced user will be able to use this form. However, for such a user the system will be more rapidly accessible, since all the information required by the command is entered at one time.

"Prompting" mode is the preferred mode for the novice user, or indeed, for the experienced user with commands that are infrequently used. In this mode, only the command name is required, although one or more parameters may additionally be supplied. Once the command processor for that command receives control, it will prompt the user for all missing parameters. For a person utilizing this mode of command invocation, the system appears to be tutorial: it is not necessary to know the order of the parameters required, or even to know which ones are required; the command processor will name the parameters it needs.

Since commands may be invoked not only by typing them at the terminal (in response to PERFORM's colon prompt), but also by placing them into Sentences or Paragraphs, and even by EXECUTE statements within INFO/BASIC programs, the choice of command invocation mode for these uses deserves some discussion. While it is possible to invoke these commands in "prompting" mode, and then to supply parameters via DATA statements, this is not a very good technique. Among the reasons that it is not are that:

(1) it requires more time, thereby slowing execution;

(2) it requires more storage space (several lines of DATA statements, instead of a single command line); and

(3) once "prompting" mode is entered, a command processor assumes real interaction with a person, so that an unexpected question could be asked for which there is no pre-stored DATA statement.

Therefore, as a general recommendation, you should use the single-line form of all commands for every invocation of a command not done by a person at a terminal.

The choice for the person at a terminal is an individual one, related to the level of experience and comfort with the system. However, in the absence of any other reason for making a choice, the performance of the overall system might be better if all users utilized the single-line mode. A command which required (for example) three parameters would begin its work after a single transmission from the terminal in the single-line mode. That first input transmission would require three additional output transmissions and three additional input transmissions in "prompting" mode. In a system with active use, the difference in throughput for all users could become significant. Note that the throughput would be influenced by more than just the number and size of the transmitted messages, but additionally by the "side effects" of potentially being "paged out" while waiting for a reply, of making other users wait for resources, etc.

```
:CREATE.FILE OLD.TRANS 3 97 'Backup of TRANS file'
Creating file OLD.TRANS / Type 3 / Modulo 97.
Creating file D_OLD.TRANS (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_OLD.TRANS.

:CLEAR.FILE EMPLOYEES
EMPLOYEES has been cleared.

:BASIC BP TEST.PROGRAM
```

FIGURE 1. Examples of single-line Command Invocation mode.

```
:CREATE.FILE
File name        =OLD.TRANS
File type        =3
Modulo           =97
File description =Backup of TRANS file
Creating file OLD.TRANS / Type 3 / Modulo 97.
Creating file D_OLD.TRANS (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_OLD.TRANS.
```

FIGURE 2. Example of "prompting" Command Invocation mode.

2.5 Commands Which Use SELECT Lists

Many PRIME INFORMATION commands that work with files or records can be invoked with an active SELECT list which comprises (as appropriate) a list of file names or record identifiers. These commands will use the active SELECT list if they are invoked in this manner.

Many of the PRIME INFORMATION commands which have either files or records as parameters of the command, will detect the presence of an active SELECT list (if there is one) and use it. A SELECT list is a list of record identifiers which has been formed by using the SELECT command (described in THE INFORM REFERENCE GUIDE, IDR39Ø5), or by using the GET.LIST or FORM.LIST commands (described in this manual). If the command being invoked expects as a parameter the name of a file (an example is the command CLEAR.FILE), you could provide a SELECT list of VOC file record identifiers that identify file records (records with "F" in field one). If the command expects as a parameter the name of one or more records in a file (such as the ED command), you could provide a SELECT list of record identifiers which are to be found in the specified file.

As a general rule for those commands which utilize a SELECT list, if there is a "file.name" parameter, but no "record.id" parameter, then you may provide a SELECT list of file identifier records which are in your account's VOC file and omit the "file.name" parameter. If the command has both a "file.name" and a "record.id" parameter, you may provide a SELECT list of identifiers of records which are in the file named "file.name", and you must specify "file.name" on the command line with the command name (but omit "record.id"). Finally, if there are parameters which either must, or optionally may, follow either "file.name" or "record.id", then those parameters (if supplied) will be used with every component of the active SELECT list: an example is HASH.TEST, where the trial file type and modulus will be used with every "file.name" in the SELECT list. However, these are general rules, and may not apply in every case. We have tried to make these commands obey logic and common sense, rather than a rigid rule: some experimentation with the command may be in order if you are not sure how a particular command will behave with a SELECT list.

BASIC	CLEAR.FILE	COPY
DELETE	DELETE.FILE	ED
ENTER	ENTRO	ENTROC
FORMAT	GROUP.STAT	GROUP.STAT.DETAIL
HASH.HELP	HASH.HELP.DETAIL	HASH.TEST
HASH.TEST.DETAIL	ISTAT	LIST
RELEASE	SORT	SPOOL
T.DUMP		

FIGURE 1. PRIME INFORMATION commands which use SELECT lists.

```
:SELECT SOURCE WITH @ID MATCHING 'PAY...'

4 records selected.

:FORMAT SOURCE

PAYMENTS.MASTER
***********************************

PAYROLL.LEDGER
**********************************************

PAYBACK.BALANCE
***********************************

PAYROLL.HISTORY
*******************************************************
```

FIGURE 2. Example of a command with a SELECT list.

2.6 Use of the BREAK Key

The BREAK key may be used to interrupt an executing process.

A process can be interrupted by pressing the BREAK key. When the BREAK key is pressed, PERFORM will interrupt the process running and display:

```
BREAK.
```

at the user terminal. The user has two options at this time:

1. Enter "G" (for "GO"), to resume the interrupted process at the point where it was interrupted.
2. Enter "Q" (for "QUIT"), to terminate the process which was interrupted and return to the PERFORM colon prompt (:).

NOTE: Carriage return, or <NEW LINE>, is not required after entering either "G" or "Q".

It is possible for a process to disable the BREAK key, as might be done to protect against interruption during a "must complete" activity such as updating several related files. To ensure proper action of the BREAK key disable/enable commands (which could be issued from several different subroutines during a process), PERFORM maintains a BREAK key counter for each user. PERFORM initializes the BREAK key counter to 0 when a process begins, and it is only when the BREAK key counter equals 0 that pressing the BREAK key will cause PERFORM to interrupt the execution of the process. Each time the BREAK key is disabled by the process, the BREAK key counter is incremented; each time the BREAK key is enabled by the process, the BREAK key counter is decremented, until it reaches 0 again. (See the BREAK KEY section in THE INFO/BASIC REFERENCE GUIDE, PDR3903, for further information on disabling/enabling the BREAK key.) Also note that the BREAK key should be used only when absolutely necessary, not as a standard operating practice. It is possible to lose output destined for the terminal by the "BREAK . . . G" sequence.

If it should become necessary to use the BREAK key to stop a process, but the BREAK key has been disabled by that process, your System Administrator has a mechanism for enabling the BREAK key for your task. See the description of the MASTER command, elsewhere in this manual.

```
:LIST STATES BY NAME

LIST STATES BY NAME 11:14:16  05-03-80  PAGE      1
STATES.... STATE............... CAPITAL........ SETTLED STATEHOOD..

AL         Alabama              Montgomery         1702 14 DEC 1819
AK         Alaska               Juneau             1784 03 JAN 1959
AZ         Arizona              Phoenix            1776 14 FEB 1912
AR         Arkansas             Little Rock        1785 15 JUN 1836
CA         California           Sacramento         1769 09 SEP 1850
CO         Colorado             Denver             1858 01 AUG 1876
CT         Connecticut          Hartford           1635 09 JAN 1788
DE         Delaware             Dover              1683 07 DEC 1787
DC         District of Columbia Washington
FL         Florida              Tallahassee        1565 03 MAR 1845
GA         Georgia              Atlanta            1733 02 JAN 1788
HI         Hawaii               Honolulu                21 AUG 1959
ID         Idaho                Boise              1842 03 JUL 1890
IL         Illinois             Springfield        1720 03 DEC 1818
<BREAK>
BREAK.
G
IN         Indiana              Indianapolis       1733 11 DEC 1816
IA         Iowa                 Des Moines         1788 28 DEC 1846
KS         Kansas               Topeka             1727 29 JAN 1861
KY         Kentucky             Frankfort          1774 01 JUN 1792
LA         Louisiana            Baton Rouge        1699 30 APR 1812
ME         Maine                Augusta            1624 15 MAR 1820
...
```

FIGURE 1. Example of use of the BREAK key.

3 PRIME INFORMATION FILES

3.1 Introduction to PRIME INFORMATION Files

PRIME INFORMATION files are designed for efficient storage and rapid retrieval of data. Each file in a PRIME INFORMATION system is defined by its own integral data dictionary, which is available for use by PERFORM, by the INFORM query and report generation language, by the commands described later in this manual, and by user-written INFO/BASIC programs.

PRIME INFORMATION files are collections of records organized as relationships. Each data file on a PRIME INFORMATION system is defined and described by its own associated dictionary, so that the dictionary is an integral part of the file. Every dictionary is defined and described by a single master dictionary, which is called DICT.DICT. Therefore, when a dictionary of a file is being modified or listed, it is treated as a data file defined by its own dictionary (DICT.DICT).

The dictionary of a data file contains several types of records which are used to describe the data portion of the file; among these are data descriptors, Information descriptors, and phrases. For a complete description of data dictionaries, see the section titled "DICTIONARIES" in THE INFORM REFERENCE GUIDE, IDR3905.

The data portion of a file is organized as a collection of logical relationships which are user-defined by the creation of the dictionary items. For example, a file named STATES might have relations such as NAME, CAPITAL, DATE.SETTLED, STATEHOOD.DATE, HIGHEST.POINT, LAND.AREA, WATER.AREA, etc. Each of these items is related to every state in the file, but the value of each relation is different for each state in the file. In this example, the record identifier for each logical relationship (i.e., for each grouping of the relations named in the example) would be something which uniquely identifies a particular state: perhaps the two-character U.S. Post Office abbreviation for the state.

Each relation in a PRIME INFORMATION file is stored as a variable-length ASCII string, delimited at least by a field mark, but possibly including other delimiters such as value marks and sub-value marks. For example, in a PERSONNEL.MASTER file, one relation might be an employee's dependents. An employee with no dependents would have a DEPENDENTS relation which contained simply a field mark (i.e., no data). An employee with a wife and two children might have a record such as:

```
Mary Jones <v> William Randolph (Billy) Jones <v> Karen Jones <f>
```

where <v> is a value mark and <f> is a field mark. Note that with this variable-length ASCII string storage technique, data storage is extremely efficient. In particular, note that the actual size of a data item is the only consideration in how much storage space is required for it. Most systems would require that the programmer reserve sufficient space in every record for an average number of dependents, and that each dependent have reserved for it an average number of character positions for the name. With PRIME INFORMATION files, an employee with no dependents requires only a single character position (a field mark) for the "dependents" relation, while our example illustrates that a very long name "fits" in the relation as well as a very short name, with no wasted space in either case.

For a further discussion of records stored as ASCII strings separated by field, value, and sub-value delimiters, see the section titled "DYNAMIC ARRAYS" in THE INFO/BASIC REFERENCE GUIDE, PDR3903.

3.2 Hash-Encoded Addressing

PRIME INFORMATION files are accessed by INFO/DMS by a method known as hash-encoded addressing, to provide rapid direct access with low time and storage overhead.

To provide rapid access to the individual relations stored in a PRIME INFORMATION file, INFO/DMS uses a method known as hash-encoded addressing. Simply stated, hash-encoded addressing is accomplished by performing a sequence of operations on the unique identifier of a record, in order to transform that unique identifier into a probable actual location within the file: i.e., into a record address. Once this probable location has been calculated, INFO/DMS can begin the search for the record at that location, rather than at the beginning of the file. Hash-encoded addressing provides rapid access by record identifier to any record in a file, but without the high storage and time cost of maintaining an index to the file.

While a complete description of hash-encoded addressing is beyond the scope of this manual (and is completely covered in many texts on computer science), a brief overview will help in understanding PRIME INFORMATION file structures. The general objective of hash-encoded addressing is to reduce the space required to store every conceivable record identifier down to the space required to store a reasonable quantity of actual records. For example, a file named STATES, with record identifiers comprised of two alphabetic characters (the U.S. Post Office official abbreviations), has a potential number of record identifiers that is 676; however, there are only 50 actual states (51, including the District of Columbia). The simplest possible hash-encoded addressing scheme would allocate 676 locations for the records in the file, with 625 of these never to be used. The waste of space, while not severe in this example, would be unaffordable if the identifier of each record were the name of the state, rather than its two-letter abbreviation. The name of the state would require a maximum of 20 characters (for the District of Columbia), and with a 26-character alphabet, this gives a potential number of records in the millions.

To reduce the mathematically calculated space requirement down to a logical space requirement, a hash-encoding algorithm is used in conjunction with a user-specified modulus. The hash-encoding algorithm typically involves performing some operation on the record identifier, whose purpose is to reduce the record identifier to a number. This number, once calculated, is usually divided by the specified modulus of the file, and the remainder from the division is used to begin the search for the record.

In the specific case of PRIME INFORMATION files, there are actually eight hash-encoded addressing algorithms employed, with the choice of any particular one effected by specifying one of the file types 2 through 9. (Type 1 files are not hash-encoded, but instead are sub-ufd type files.) In general, the hash-encoded addressing algorithms employed have in common that they process each character of the record identifier one-by-one, accumulating its algorithm-determined value into a 32-bit word.

Once the 32-bit number has been calculated by the hash-encoded addressing algorithm, it is divided by the modulus defined for the file. The remainder from this division is used as a group number, to identify the location of the group of records where the relation identified by the record identifier will reside. The number of such groups is actually the modulus of the file; e.g., a file with a modulus of seven will have all of its records organized into seven groups.

With a careful choice of file type and modulus, and a logical choice for record identifier which assures that each record identifier is sufficiently distinct from each other record identifier, the user has considerable control over the resulting distribution of records across the total number of groups. The objective of exercising these choices is to arrive at as even a distribution of records across all groups as is possible, while keeping each group size relatively small. When a file has been created and data placed into it, the GROUP.STAT and GROUP.STAT.DETAIL commands may be used to analyze this record distribution. (See the descriptions of these commands elsewhere in this manual.)

For further details on hash-encoded addressing techniques and algorithms, there are many excellent books. One particularly outstanding one is Volume 3, "Sorting and Searching", in Donald E. Knuth's series "The Art of Computer Programming", published by Addison-Wesley.

3.3 PRIME INFORMATION File Groups

Records in PRIME INFORMATION files are stored in physical groups; the group that a particular record is stored in is determined by hash-encoding the record identifier (key).

The groups which are used to store logical records of PRIME INFORMATION files are physical disk records, or strings of physical disk records, organized as either SAM or DAM sub-files of segmented files (SEGSAM files or SEGDAM files; see the section titled "Large Files" for an explanation of how the choice is made between SEGSAM and SEGDAM files.) Each physical disk record contains 2080 bytes, of which 32 bytes are used by the system for disk management and data integrity, leaving 2048 bytes available for data storage. This 2048 bytes is further diminished by INFO/DMS, the data base manager of PRIME INFORMATION. The exact amount of space required by INFO/DMS to store logical records in a group is dependent on several factors, such as the size of each key, the size of the data portion of each record in the group, the number of logical records in the group, and certain other variables which are outside the scope of this manual. A formula which is accurate (but not precise, since it does not account for all possible variables) is:

```
     key.length + (data.length * 1.11) + 4
```

where "data.length" and "key.length" are expressed in characters and are rounded upward to be evenly divisible by two. This formula must be repeated for each record in a group, and the total must have an additional twelve characters added to it (for group identifier overhead).

When a file is created (using the CREATE.FILE command, described elsewhere in this manual), the modulus parameter is used to determine the number of groups that will comprise the file. If the file type and modulus have been chosen well for the size and type of data to be stored in the file, then when the file is fully loaded with data, there will be exactly one disk record for each group of logical records in the file. If the modulus is too small for the quantity of data, then each group in the file will be considerably larger than 2048 bytes, thus requiring that each group be a chain of physical records rather than a single physical record. The number of physical disk records which must be read to retrieve a particular logical record is expressed as the total number of physical disk records which comprise the group, divided by two, and rounded upward to the nearest whole number. Thus, a group with only a single physical disk record never requires more than a single physical disk access to retrieve any logical record of the group, while groups with three disk records will, on average, require two disk accesses. Obviously, the performance of the system will be less than it could be when accessing such a file.

If the modulus is too large for the quantity of data, then there will be many groups which are "empty" (i.e., have no logical records in those groups). Although this is also an undesirable situation, it is better than having groups that are too large. PRIME INFORMATION files are allocated as "sparse" files: that is, a group is not created unless and until there is data to be placed in the group. Therefore, a file with a large number of empty groups will not take extra space to store, nor extra time to access, because it has empty groups. Nevertheless, the objective is to have the logical records of your PRIME INFORMATION files as evenly distributed across the groups as is possible, with none empty, and all approximately the same size. Several tools are provided with the PRIME INFORMATION system to monitor logical record distribution within PRIME INFORMATION files (see the sections on the GROUP.STAT and GROUP.STAT.DETAIL commands in this manual), and to reorganize these files when necessary (see the sections on the RESIZE, HASH.TEST, and HASH.TEST.DETAIL commands in this manual).

As a general rule of thumb, a file is well-hashed if each group in the file is approximately the same size as the others, and if no group in the file exceeds approximately 1900 bytes in size. Since poorly-hashed files are a major causative factor in poor system performance, group size and record distribution across groups should be closely examined if performance begins to deteriorate for no apparent reason.

Note that INFO/DMS automatically deletes physical disk records within groups when those records are no longer needed (when logical records within those groups are deleted, or are diminished in size). However, once a group is created (by placing a logical record in that group), at least one physical disk record is retained for that group, even if the entire group becomes empty. The RESIZE command, described in this manual, may be used to delete all disk records associated with empty groups.

3.4 File Types

The PRIME INFORMATION system provides eight file types, types 2 through 9, each of which is associated with its own hash-encoded addressing technique. In addition, file type 1 is provided, as a sub-ufd type file which does not use hash-encoded addressing.

The PRIME INFORMATION system provides eight file types which are used for hash-encoded direct access, and one file type which is a PRIMOS sub-ufd file type. The PRIMOS sub-ufd file type is a Type 1 file, and is used for INFO/BASIC program source and object code, and for files destined for interchange with PRIMOS-level processors (such as RUNOFF). File types 2 through 9 are the hash-encoded direct access files, and are the principal data files used on PRIME INFORMATION systems.

Each of these file types is described in more detail in succeeding sections of this manual.

FILE TYPE	PRINCIPAL USE
1	INFO/BASIC source code storage Data interchange (via SAM file) to PRIMOS
2	Data files with record identifiers that are more unique in right-most characters than in left-most characters; for general use
3	Data files with record identifiers that are principally numeric, but with common separators (such as *, -, or #); more unique in right-most characters than in left-most characters.
4	Data files with record identifiers that are alphabetic (comprised of the 64-character ASCII subset); more unique in right-most characters than in left-most characters.
5	Data files with records identifiers that span the entire ASCII character set; more unique in right-most characters than in left-most characters.
6	Data files with record identifiers like those of Type 2 files, but more unique in their left-most character positions than in their right-most characters
7	Data files with record identifiers like those of Type 3 files, but more unique in their left-most character positions than in their right-most characters
8	Data files with record identifiers like those of Type 4 files, but more unique in their left-most character positions than in their right-most characters
9	Data files with record identifiers like those of Type 5 files, but more unique in their left-most character positions than in their right-most characters

FIGURE 1. Summary of the nine PRIME INFORMATION file types.

3.5 File Type 1 Description

The PRIME INFORMATION Type 1 file is intended for use in storing INFO/BASIC program source and compiled object code, and for providing a data interchange path with PRIMOS-level processors such as RUNOFF.

The PRIME INFORMATION Type 1 file is implemented physically as a PRIMOS sub-ufd. To the PRIME INFORMATION processors, the Type 1 file appears to be a named file which contains records that have unique record identifiers. For example, a Type 1 INFO/BASIC source and object code file named BP might have records (programs) named UPDATE.CUST.MASTER, ORDER.ENTRY, and INVENTORY.UPDATE. This same Type 1 file can be viewed by PRIMOS as a sub-ufd to the PRIME INFORMATION account, where the name of the sub-ufd is BP, and the contents of the sub-ufd are the three PRIMOS SAM files named UPDATE.CUST.MASTER, ORDER.ENTRY, and INVENTORY.UPDATE.

INFO/BASIC source and object code (and any files to be $INSERTed during compilation) must be stored in Type 1 files; file Types 2 through 9 are not usable for this purpose. Note that since both PRIME INFORMATION and PRIMOS recognize Type 1 files (although they each have a different "view" of these files as described above), data stored in Type 1 files is available to processors at both levels. In the particular case of INFO/BASIC source code, both the PRIME INFORMATION EDITOR and the PRIMOS EDITOR can operate on the source code. Additionally, the BASIC compiler can compile these programs either from the PRIME INFORMATION level, or from the PRIMOS level.

There are other processors which have an equivalent in both environments, and which may utilize the dual-environment character of Type 1 files to advantage. In particular, Prime's text processing system (RUNOFF) can be used to advantage by creating the source files within the PRIME INFORMATION environment (as records in a Type 1 file), then invoking RUNOFF in the PRIMOS environment.

It may be desirable to provide a data transfer mechanism from the data stored in PRIME INFORMATION hash-encoded direct access files, to other language processors which run only in the PRIMOS environment. For example, it would be possible to write a simple INFO/BASIC program which reads a direct access (Type 2 through 9) file, formats it into fixed-length fields padded with trailing spaces or leading zeroes as appropriate, and writes records into a Type 1 (sub-ufd) file. Each "record" written by the INFO/BASIC program becomes a SAM file, which PRIMOS-level programs written in (for example) COBOL or FORTRAN can process. The inverse operation is also possible, of course: a COBOL or FORTRAN program could create a SAM file in a PRIME INFORMATION account's sub-ufd (where that sub-ufd was created by the PRIME INFORMATION account as a Type 1 file), and an INFO/BASIC program could transfer that data into the direct access data file Types 2 through 9.

Note that when a Type 1 file is created, the modulo parameter required for Types 2 through 9 is not used. This is because a Type 1 file is not a hash-encoded direct access type file, but a sub-ufd; the modulo parameter is used only for the hash-coding function, which is not used for Type 1 files.

For additional documentation on PRIMOS sub-ufd's and SAM files, see the section titled "File Management System Concepts" in the PRIMOS Subroutines Reference Guide, PDR3621.

3.6 File Type 2 Description

The PRIME INFORMATION Type 2 file is a hash-encoded direct access data file that uses a general-purpose hash-encoding algorithm.

The PRIME INFORMATION Type 2 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is a general-purpose algorithm, suitable for most data files whose record identifiers are relatively unique in the right-most portion of each identifier.

In a Type 2 file, each record identifier is treated as if it were a numeric identifier, even though it may contain non-numeric characters. This file type is ideal for records whose record identifiers are sequential numbers, because of the assumption of a numeric identifier and because only the right-most eight characters of the record identifier are used in the hash-encoding algorithm (since the right-most eight characters of the number are the most unique portion of each sequential number).

To prepare the record identifier for division by the modulo selected for the file, the hash-encoding algorithm begins by extracting from the record identifier the right-most eight characters (or the entire record identifier is used, if it is less than eight characters long). Next, a 32-bit accumulator is initialized to zero, so that successive characters of the record identifier may be added to it.

Each character of the extracted (up to eight-character) record identifier is treated individually by the hash-encoding algorithm. The assumption that the record identifier is numeric is enforced by truncating the left-most four bits of each character, and using only the right-most four bits in the algorithm. These low-order four bits can only have a value between zero and fifteen, regardless of the ASCII value of the character.

Proceeding character-by-character from the left-most (high-order) position of the extracted record identifier, the algorithm operates as follows:

1. Multiply the current value of the accumulator by 10. (This provides proper scaling for a decimal value added digit by digit.)

2. Add the next character's low-order four bits to the accumulator.

3. Repeat steps 1 and 2 for each character in the extracted record identifier, to a maximum of eight characters.

The net result of this algorithm is readily understood with an example. If a record identifier with a length of nine characters, and an ASCII value of "123456789" is processed by this algorithm, the extracted record identifier would be the right-most eight characters, in this case "23456789". PRIME's ASCII representation of the digits 0 through 9 is the octal values from 260 through 271. The low-order (right-most) four bits of these octal values is the binary equivalent of 0 through 9. As each of these digits is added to the accumulator, with scaling by 10 for each addition, the end result in the accumulator is the binary equivalent of the extracted record identifier: "23456789".

The Type 2 file, because of its general-purpose hash-encoding algorithm, can be used successfully in most cases. However, note that its operation provides even distribution of the records only when they are relatively unique in their right-most eight character positions. For files whose record identifiers are non-unique in their right-most eight character positions, but are relatively unique in their left-most eight character positions, there is a "left-weighted" equivalent of this file type in the Type 6 file, described elsewhere in this manual. For other cases, see the descriptions of the other six direct access file types in this manual.

3.7 File Type 3 Description

The PRIME INFORMATION Type 3 file is a hash-encoded direct access data file which employs a hash-encoding algorithm similar to the one used for Type 2 files, but which results in better distribution of records that have certain recurring characters within their record identifiers.

The PRIME INFORMATION Type 3 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is similar to the algorithm used for Type 2 files. However, Type 3 files are particularly useful where the record identifiers are comprised of data that is principally numeric, but with common separators between logical components of that data.

As an example, a file whose record identifiers are Social Security numbers has a record identifier for each record which looks like this:

    nnn-nn-nnnn

where each "n" is a numeric digit. With a Type 2 file, the right-most eight characters of this eleven-character record identifier would be used in the hash-encoding algorithm. In this example, every record identifier would have two "-" characters involved equally in the hash-encoding algorithm, even though these are insignificant characters in terms of the data (they are merely separator characters). The following description of the hash-encoding algorithm used for Type 3 files will illustrate the usefulness of this file type for record identifiers such as this.

The Type 3 file hash-encoding algorithm functions identically to the Type 2 hash-encoding algorithm, with the following exception: instead of extracting the right-most eight characters of the record identifier for use in the algorithm, the right-most eight <u>numeric</u> characters are used. If a character is encountered in the right-most eight characters of the record identifier, whose right-most four bits are not one of the digits 0 through 9, it is passed over for the next character in the record identifier. This process is continued until there are eight numeric digits in the extracted record identifier, or until there are no more characters in the original record identifier, whichever comes first.

The net effect of using only the truly numeric characters from the record identifier is that common separator characters used in otherwise numeric identifiers are "thrown away", and do not create a bias in the distribution of the records within their groups. Since many applications have record identifiers which are principally numeric, but contain separators such as "*", "-", "/", ".", etc., using a Type 3 file for these applications results in a more uniform distribution of records than would the Type 2 file.

As in the case of the Type 2 file, the Type 3 file has its logical inverse: for record identifiers which are more unique in their first eight characters than in their last eight characters, the Type 7 file should be used.

Note that use of the Type 3 file for record identifiers which contain no numeric characters whatsoever, will result in placing most records of the file into a single group. While a Type 2 file merely treats every character of the record identifier as if it were numeric, the Type 3 file requires that at least some characters in the right-most eight characters of every record identifier be numeric. If this requirement is not met, the distribution of records into the groups of the file will be very poor, and will adversely affect the performance of the system.

Note also that, for the sake of efficiency, the test for whether a character is or is not numeric is based solely on its right-most four bits. That is, if the right-most four bits of a character are in the range of Ø through 9, that character is treated as if it were numeric (for purposes of hash-encoding), even though many characters of the ASCII character set that are not numeric will satisfy this test.

3.8 File Type 4 Description

The PRIME INFORMATION Type 4 file is a hash-encoded direct access data file which uses a hash-encoding algorithm weighted toward alphabetic record identifiers.

The PRIME INFORMATION Type 4 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is weighted toward record identifiers comprised of alphabetic characters, and is therefore the preferred file type for records with this type of record identifier.

To prepare the record identifier for division by the modulo selected for the file, the hash-encoding algorithm begins by extracting from the record identifier the right-most five characters (or the entire record identifier is used, if it is less than five characters long). Next, a 32-bit accumulator is initialized to zero, so that successive characters of the record identifier may be added to it.

Each character of the extracted (up to five-character) record identifier is treated individually by the hash-encoding algorithm. The assumption that the record identifier is alphabetic is enforced by truncating the left-most two bits of each character, and using only the right-most six bits in the algorithm. These low-order six bits can have a value between zero and sixty-three, a range which corresponds directly to the 64-character ASCII printable character set.

Proceeding character-by-character from the left-most (high-order) position of the extracted record identifier, the algorithm operates as follows:

1. Multiply the current value of the accumulator by 64. (This provides proper scaling for an alphabetic value added character by character.)

2. Add the next character's low-order six bits to the accumulator.

3. Repeat steps 1 and 2 for each character in the extracted record identifier, to a maximum of five characters.

The Type 4 file, because of its alphabetic-weighted hash-encoding algorithm, can be used successfully for most files with alphabetic record identifiers. However, note that its operation provides even distribution of the records only when they are relatively unique in their right-most five character positions. For files whose record identifiers are non-unique in their right-most five character positions, but are relatively unique in their left-most five character positions, there is a "left-weighted" equivalent of this file type in the Type 8 file, described elsewhere in this manual. For other cases, see the descriptions of the other six direct access file types in this manual.

3.9 File Type 5 Description

The PRIME INFORMATION Type 5 file is a hash-encoded direct access data file which employs a hash-encoding algorithm that is unweighted by character type. It assumes equal probability for all ASCII character types as meaningful characters in each character position of the record identifier.

The PRIME INFORMATION Type 5 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is not weighted toward either numeric or alphabetic record identifiers; it gives equal weight to any ASCII character which appears in the record identifier. This algorithm is useful for files whose record identifiers may contain any character from the 64-character printing ASCII set.

To prepare the record identifier for division by the modulo selected for the file, the hash-encoding algorithm begins by extracting from the record identifier the right-most four characters (or the entire record identifier is used, if it is less than four characters long). Next, a 32-bit accumulator is initialized to zero, so that successive characters of the record identifier may be added to it.

Each character of the extracted (up to four-character) record identifier is treated individually by the hash-encoding algorithm. Since each character in the record identifier is equally weighted with each other character, no bits are extracted from the character: the entire eight bits are used.

Proceeding character-by-character from the left-most (high-order) position of the extracted record identifier, the algorithm operates as follows:

1. Shift the current contents of the accumulator to the left by eight bits (to make room for the next eight-bit character).

2. Move the next character's full eight bits to the accumulator's right-most eight bits.

3. Repeat steps 1 and 2 for each character in the extracted record identifier, to a maximum of four characters.

The Type 5 file, because of its unweighted hash-encoding algorithm, can be used successfully for almost any record identifiers. However, note that its operation provides even distribution of the records only when they are relatively unique in their right-most four character positions. For files whose record identifiers are non-unique in their right-most four character positions, but are relatively unique in their left-most four character positions, there is a "left-weighted" equivalent of this file type in the Type 9 file, described elsewhere in this manual. For other cases, see the descriptions of the other six direct access file types in this manual.

3.10 File Type 6 Description

The PRIME INFORMATION Type 6 file is a hash-encoded direct access data file which employs a hash-encoding algorithm identical to the one used for Type 2 files, but using the left-most eight characters of the record identifier instead of the right-most eight characters.

The PRIME INFORMATION Type 6 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is the complement of the algorithm employed with Type 2 files, in that the left-most eight characters of each record identifier are extracted for use in the algorithm instead of the right-most eight characters.

Thus, Type 6 files should be employed where the record identifiers are more unique in their high-order (left-most) character positions than in their low-order (right-most) positions. With this exception, the detailed description of the Type 2 file hash-encoding algorithm applies equally to the Type 6 algorithm, and is therefore not repeated here.

3.11 File Type 7 Description

The PRIME INFORMATION Type 7 file is a hash-encoded direct access data file which employs a hash-encoding algorithm identical to the one used for Type 3 files, but using the left-most eight characters of the record identifier instead of the right-most eight characters.

The PRIME INFORMATION Type 7 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is the complement of the algorithm employed with Type 3 files, in that the left-most eight characters of each record identifier are extracted for use in the algorithm instead of the right-most eight characters.

Thus, Type 7 files should be employed where the record identifiers are more unique in their high-order (left-most) character positions than in their low-order (right-most) positions. With this exception, the detailed description of the Type 3 file hash-encoding algorithm applies equally to the Type 7 algorithm, and is therefore not repeated here.

3.12 File Type 8 Description

The PRIME INFORMATION Type 8 file is a hash-encoded direct access data file which employs a hash-encoding algorithm identical to the one used for Type 4 files, but using the left-most five characters of the record identifier instead of the right-most five characters.

The PRIME INFORMATION Type 8 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is the complement of the algorithm employed with Type 4 files, in that the left-most five characters of each record identifier are extracted for use in the algorithm instead of the right-most five characters.

Thus, Type 8 files should be employed where the record identifiers are more unique in their high-order (left-most) character positions than in their low-order (right-most) positions. With this exception, the detailed description of the Type 4 file hash-encoding algorithm applies equally to the Type 8 algorithm, and is therefore not repeated here.

3.13 File Type 9 Description

The PRIME INFORMATION Type 9 file is a hash-encoded direct access data file which employs a hash-encoding algorithm identical to the one used for Type 5 files, but using the left-most four characters of the record identifier instead of the right-most four characters.

The PRIME INFORMATION Type 9 file is one of the eight data file types which provide hash-encoded direct access to each record in the file. The hash-encoding algorithm employed to provide this direct access is the complement of the algorithm employed with Type 5 files, in that the left-most four characters of each record identifier are extracted for use in the algorithm instead of the right-most four characters.

Thus, Type 9 files should be employed where the record identifiers are more unique in their high-order (left-most) character positions than in their low-order (right-most) positions. With this exception, the detailed description of the Type 5 file hash-encoding algorithm applies equally to the Type 9 algorithm, and is therefore not repeated here.

3.14 Large Files

Unusually large files on the PRIME INFORMATION system are optimized for efficient access automatically when the file is created. The signal to CREATE.FILE that this optimization should take place is the selection of a file modulo that is 2000 or larger.

Any of the hash-coded direct access files (types 2 through 9, which require the specification of a modulus at the time the file is created) can be optimized for the efficient storage and rapid retrieval of a very large quantity of records. This is done by CREATE.FILE automatically, by the choice of the physical file type used to implement the file. CREATE.FILE normally uses the PRIMOS Segmented Sequential Access Method (SEGSAM) file type to implement hash-encoded direct access files. However, when the file is created with a modulus of 2000 or greater, CREATE.FILE uses the PRIMOS Segmented Direct Access Method (SEGDAM) file type to implement the file.

Regardless of which physical file type is used by CREATE.FILE to implement the hash-encoded direct access file, all file access by user-written INFO/BASIC programs is the same for all of these file types, and all of the PRIME INFORMATION processors function identically regardless of physical file type. However, for very large quantities of records, the SEGDAM type physical file is more efficient than the SEGSAM file type, so INFO/DMS chooses the SEGDAM type when the chosen modulus (2000 or greater) indicates that the number of records to be stored in the file is large. The converse is also true: for files of average size, SEGSAM physical files are more efficient than SEGDAM file types, so CREATE.FILE uses the SEGSAM file structure to implement hash-encoded direct access files for most files created on the system.

It should be noted that physical file type (SEGSAM or SEGDAM) is determined only at the time the file is created. Specifically, when a file is created with a modulus of (for example) 379, the SEGSAM file type is chosen. If this file is reorganized with the RESIZE command (described elsewhere in this manual) using a new modulus which is greater than 2000, the physical file type remains SEGSAM; RESIZE does not change physical file type.

For additional information on SEGSAM and SEGDAM file types, see the section titled "File Management System Concepts" in the PRIMOS SUBROUTINES REFERENCE GUIDE, PDR3621.

4 THE VOCABULARY FILE

4.1 The User Vocabulary File

Each PRIME INFORMATION account has a unique vocabulary that can be tailored to the user's needs. This unique vocabulary is stored in the VOC file of the user's account.

A user's vocabulary consists of defining records for verbs, file names, keywords, menus, stored sentences, and paragraphs. This vocabulary is unique to each PRIME INFORMATION account, thus allowing vocabularies to be tailored to each user's or each application's needs. The provision for a private vocabulary for each account on the system allows control over the capabilities of each PRIME INFORMATION account, by adding or removing vocabulary definitions of items such as verbs and keywords.

Each PRIME INFORMATION account's vocabulary is stored in a file within that account; the name of this file is VOC. The VOC file is created automatically as part of the process which creates PRIME INFORMATION accounts. The portion of that process which creates the VOC file is accomplished by copying all of the records of the file named NEWACC (in the master PRIME INFORMATION account named ISYS) to the VOC file of the account being created. A standard NEWACC file is supplied with the system. The installation administrator can add or delete items in NEWACC to fit the installation's particular needs, so that the creation of a new PRIME INFORMATION account on the system results in a standard VOC file. After this standard VOC file has been created in the new user's account, the system administrator can modify it by either adding or deleting entries, as desired.

A second file is also established in each user account, named VOCLIB. It is recommended that long paragraphs be stored in this file, using remote records. Using the VOCLIB in this manner keeps the size of the VOC file smaller, and allows more rapid access to the items in the VOC file. (See the section on REMOTE DEFINITION in this manual.)

4.2 VOC File Record Types

A PRIME INFORMATION account's vocabulary is defined in its VOC file. Each record in the VOC file has a type code in field one, which is used by PERFORM to identify the type of the item defined by the record.

The VOC file contains records defining verbs, stored sentences, menus, paragraphs, phrases, files, keywords, and remote file references. Each record contains a type code in field one, which defines to PERFORM which of several types the particular record is. The type codes recognized by PERFORM are:

V	Verbs
S	Stored Sentences
M	Menu Pointers
PA	Paragraphs
PH	Phrases
F	File Pointers
K	Keywords
I	Information Descriptors
R	Remote items
X	User-defined (ignored by PRIME INFORMATION software)

A reserved type code of X is provided for users who may wish to store something in the VOC file which does not belong to any of the categories listed above. Note that this practice is not recommended, but is allowed. (See the section titled "USER Record Definition" in this manual.)

Each of these VOC file record types is described in more detail in the following sections of this manual, except that type code "I" (for Information Descriptors) is described in THE INFORM REFERENCE GUIDE, IDR3905.

V	Verbs
S	Stored Sentences
M	Menu Pointers
PA	Paragraphs
PH	Phrases
F	File Pointers
K	Keywords
I	Information Descriptors
R	Remote items
X	User-defined (ignored by PRIME INFORMATION software)

FIGURE 1. Type Codes in the VOC File.

4.3 SENTENCE Definition Record

Stored sentences are VOC file records which contain sentences that may be executed by typing a single word name.

A sentence that is expected to be used more than once may be stored in the VOC file with a single word name as its key. Once stored, the sentence may be executed again by simply typing the name of the sentence at the PERFORM colon prompt (:), or by invoking the sentence by its name from within a paragraph. (See the section titled "PARAGRAPH Definition Record" in this manual.) Stored sentence records in the vocabulary file contain an "S" as the first character in field one, with an optional description of the purpose of the sentence in the remainder of field one. Field two is the sentence that will be executed when the name of the sentence is encountered by PERFORM.

When the first word in a command is the name (record identifier) of a stored sentence, the list of words in field two of the record replaces the first word in the command. The command in its new form is then re-analyzed by PERFORM. Note that all other words in the original command are left unchanged. Thus it is possible to append words to a stored sentence and have PERFORM treat it as if the entire stored sentence and the appended words were all typed at the colon prompt.

Stored sentences are useful for situations where a sentence is long and complex and is going to be used more than once, or where any sentence is going to be used frequently and it is desired to reduce the amount of typing or the possibility of typing errors. One example of this would be an INFORM report-generating sentence that is to be produced periodically, and the INFORM sentence is complex or very long. This sentence could be stored in the vocabulary file, allowing that report to be referred to and generated by a single word name.

If the sentence is too long to fit on a single line, it may be continued onto successive lines by typing an underscore character (_) at the end of each line except the last line of the sentence. (The underscore character may be a back-arrow character on some terminals.)

See the section titled "PERFORM SENTENCE STACK" in this manual for a convenient way to create stored sentences using the ".S" command, or to modify them using the ".R name" command.

The stored sentence LISTS displays all stored sentence definition records in the VOC file.

```
 id:  name of stored sentence

001:  S {description}
002:  the text of the stored sentence
```

FIGURE 1. Format of the Stored SENTENCE Record.

Given the VOC file record:

```
 id:  REPORT.3

001:  S WEEKLY REPORT FOR INVENTORY REORDERING
002:  LIST INV WITH QOH < 20 DESC QOH COST
```

the command:

```
:REPORT.3 LPTR
```

is the same as the command:

```
:LIST INV WITH QOH < 20 DESC QOH COST LPTR
```

FIGURE 2. Example of a Stored Sentence.

4.4 PARAGRAPH Definition Record

A series of PERFORM sentences to be executed in sequence may be stored in the VOC file as a named paragraph, for subsequent execution by use of a single word command.

Paragraphs are a series of PERFORM sentences kept in a single record in the VOC file, for subsequent execution by simply typing the name of the paragraph. The first two characters of field one of the paragraph record must be the letters "PA", to identify this as a paragraph record; a description of the paragraph may optionally be included in field one. Each of the remaining fields contains a PERFORM sentence, using the same format as would be used at the terminal keyboard. (See the section titled "SENTENCE Definition Record" in this manual.)

Sentences in a paragraph are executed in sequence. Each sentence is analyzed by PERFORM just as if it had been entered at the terminal keyboard. Therefore, commands in the paragraph may be complete sentences, or they may be names of stored sentences, names of paragraphs, names of menus, etc.

When the paragraph has been completed (i.e., when there are no more commands to be executed), control returns to PERFORM's colon prompt or, if the paragraph was initiated from a menu or paragraph, to that menu or paragraph.

If the sentence is too long to fit on a single line, it may be continued onto successive lines by typing an underscore character (_) at the end of each line except the last line of the sentence. (The underscore character may be a back-arrow character on some terminals.)

The stored sentence LISTPA displays all paragraph definition records stored in the VOC file.

A note of warning: it is possible, either deliberately or inadvertently, to create a paragraph which invokes itself. If this happens, the paragraph will never complete, and the task will have to be aborted (with the BREAK key, described elsewhere in this manual). This inadvertent recursion is most likely to happen when a paragraph invokes another paragraph, which invokes another, etc., until one of the paragraphs in the recursion string invokes a paragraph which has been invoked earlier.

```
 id:  name of the paragraph

001:  PA {description}
002:  first sentence
003:  next sentence
004:  next sentence
  .
  .
  .
nnn:  last sentence
```

FIGURE 1. Format of the PARAGRAPH Record.

```
 id:  INVENTORY.RUN.1

001:  PA NIGHTLY INVENTORY AUDIT
002:  LIST INVENTORY.FILE LPTR
003:  SELECT INVENTORY.FILE BY CUST.NUMBER
004:  RUN BP INV0237A
005:  LIST ERROR.FILE LPTR
```

FIGURE 2. Example of a Paragraph.

4.5 PHRASE Definition Record

A PHRASE is a fragment of an INFORM sentence which is stored as a record in the VOC file or in a data file dictionary. A phrase may be included in any sentence by typing the name of the phrase into the sentence.

A PHRASE is an incomplete fragment of a sentence, stored as a record in the VOC file or in a data file dictionary. The record identifier of a phrase is the name of the phrase, chosen by the person who created it. A PHRASE cannot contain a VERB name; since a SENTENCE must begin with a VERB, a phrase can never be a complete sentence. (See the section titled "SENTENCE Definition Record" in this manual.)

A phrase can be included in a sentence at any point after the VERB name, by simply putting the name of the phrase into the sentence at the appropriate point. PERFORM will find the named phrase record either in the dictionary of the file named in the sentence, or in the VOC file. Once found, the actual phrase is extracted from the record and inserted into the sentence in place of the phrase name, as if the phrase itself had been typed into the sentence.

A phrase record in the VOC file or in a data file dictionary must contain the letters PH as the first two characters in field one, to identify the record as a stored phrase. An optional description may also be included in field one, after the PH. Field two contains the list of words that constitute the phrase that will replace the name of the phrase in the INFORM sentence.

Phrases may be kept in the VOC file, or in the dictionary of the file named in the sentence. When PERFORM encounters the name of a phrase in a sentence, it will search the named file dictionary for the phrase before it searches the VOC file. Therefore, if a phrase is stored in a data file dictionary and in the VOC file with an identical name in both locations, the phrase from the data file dictionary is the one that is used by PERFORM to complete the sentence.

The stored sentence LISTPH displays all phrase definition records stored in the VOC file.

```
 id: phrase.name

001: PH {description}
002: list of words that replace "phrase.name"
     in the INFORM sentence
```

FIGURE 1. Format of PHRASE Records.

Given a phrase record of:

```
 id: BACK.ORDERS

001: PH
002: WITH BO.QTY > 0 CUST BO.QTY SH.DATE
```

then the command:

```
:LIST ORDER.FILE BACK.ORDERS LPTR
```

is the same as entering the command:

```
:LIST ORDER.FILE WITH BO.QTY > 0 CUST BO.QTY SH.DATE LPTR
```

FIGURE 2. Example of the use of a Phrase.

4.6 MENU Pointer Record

A MENU is a list of processes from which a selection may be made for execution by PERFORM. A MENU is defined by a record kept in a MENU file, and it is invoked by specifying the name of a MENU pointer record that is kept in the VOC file.

A MENU is a list of processes from which a selection may be made for execution. Each selection is identified by a brief description of an operation, chosen by the person who created the MENU. Associated with each description is a selection number, which was assigned by the MENU processor when the MENU was invoked. When a selection of one of the described processes is made (by typing its associated number in response to a prompt from the MENU processor), a sequence of PERFORM commands that was defined by the creator of the MENU and associated with the selected process description, is executed. After completion of a selected process, the MENU's selections are again displayed so that another selection may be made.

This sequence of events (displaying a narrative description of potential operations, selecting and executing a particular operation, then displaying potential operations again) is repeated until no choice is made (usually signified by <NEW LINE> without any selection number for most MENUs; see the description of MENU "exit" and "stop", in the section titled "MENU Definition Records" in this manual, for other means of exiting a MENU.) Upon exit from a MENU, control is returned to the entity that invoked the MENU, which could be PERFORM (signified by a colon prompt after exiting the MENU), or could be another MENU (signified by a re-display of the choices of the invoking MENU), or could be a PARAGRAPH or program, etc.

The MENU pointer record in the VOC file contains an "M" in field one to identify it as a MENU pointer record; a "comment" describing the MENU may optionally follow the "M". Field two contains the name of the file where the MENU is stored, and field three contains the record identifier (the name) of the actual MENU definition record; this MENU definition record is described in the section of this manual titled "MENU Definition Record".

A MENU maintenance processor which allows the user to create or modify a MENU, list the contents of a particular MENU, list all MENUs stored in a particular MENU file, or perform other MENU-related functions, is provided with the system. This processor, itself MENU-driven, is invoked by entering the command "MENUS" in response to the colon prompt from PERFORM. This MENU maintenance processor is described in more detail in the section of this manual titled "MENUS: A MENU Maintenance Processor".

```
Menu Pointer in VOC file

 id: MENU name

001: M {description}
002: file.name of file where the MENU record
     is to be found
003: id of record in file (named in field two
     above) that is the actual MENU record
```

FIGURE 1. Format of the MENU pointer record in the VOC file.

4.6.1 MENU Definition Records

A MENU definition record in a MENU file defines the screen format of a MENU and the processes that are to be executed when a selection is made from that MENU. Titles and alternate query prompts, exit commands, and stop commands are also defined in a record in a MENU file.

A MENU definition record in a MENU file defines the screen format of a MENU and the processes that are to be executed when a selection is made from that MENU. This MENU definition record can be built using the system-supplied PRIME INFORMATION MENU maintenance processor "MENUS", described in the section of this manual titled "MENUS: A MENU Maintenance Processor". Once a MENU definition record has been put into a MENU file, a MENU pointer record must be put into the VOC file in order to invoke that MENU (see the section titled "MENU Pointer Record" elsewhere in this manual).

The actual MENU definition record contains seven fields, and this record is stored either in a user-created MENU file which is available for use only within that account (the usual case), or in the system MENU file named "ISYS.MENU.FILE", which is available to all users on the system. Field one of this record is used to store the (optional) title of the MENU, and is a singled-valued field which may contain a text string of from 1 to 55 characters. If a text string is present in field one, then that text is used as the MENU title, and is displayed, with the current system time and date, on the first line of the MENU display screen. (The procedure for positioning a title elsewhere on the screen and/or to override the display of the system time and date with the title is described later in this section.) The title will be centered by PERFORM when the MENU is displayed, according to the number of characters in the text string; a title that is greater than 55 characters long will be truncated at the 55th character.

Field two is a multi-valued field, with each value being a description of a process which may be selected when the MENU is displayed. Selection numbers (consecutive integers starting with one for the first process) are assigned to each process description when the MENU is displayed. Each process description, preceded by its assigned selection number, is displayed on successive lines of the user's terminal, beginning on line 3, and always beginning in position 1 of each line. When the MENU is created, it may optionally be created as a formatted MENU, which allows the MENU's creator to specify where each selection should be displayed on the terminal's screen. This is accomplished by entering the description text in the following format:

```
(COL,ROW)description.text
```

where COL and ROW are the column and row at which the "description.text" will be displayed on the screen. The "description.text" must be entered in the exact format as shown, for the cursor-positioning function of a formatted MENU to work properly.

Field three is multi-valued, and is associated value-by-value with the values in field two; it is a list of PERFORM commands that will be executed if the corresponding process description is selected. For example, if process number four is selected (the fourth value of field two), then command number four is executed (the fourth value of field three). There are four types of commands that are acceptable in this "action" field. The four command types are:

(1) A sentence which begins with a Verb;

(2) The name of a stored Sentence;

(3) The name of a Paragraph; or

(4) The name of a MENU Pointer.

These four command types are each described in detail elsewhere in this manual. If the action is a valid PERFORM sentence, then this sentence will be executed; after it has been completed, the MENU will be re-displayed, and the user will be given another opportunity to make a selection. If the action is the name of a stored Sentence, then the sentence defined by this name will be executed; after it has been completed, the MENU will be re-displayed, and the user will be given another opportunity to make a selection. If the action to be executed is the name of a Paragraph, then all commands in that Paragraph will be executed; after it has been completed, the MENU will be re-displayed, and the user will be given another opportunity to make a selection. If the action to be executed is the name of a MENU pointer, then control will be passed to the MENU referenced and the user will be prompted to make a selection from the new MENU. When the user no longer desires to execute any commands in the new MENU, this can be signified by pressing <NEW LINE> without having typed a selection number (or, for some MENUs, by typing one of the defined "exit" or "stop" commands; "exit" and "stop" commands will be discussed later in this section).

If one of the action values of field three is left null, then the corresponding description value in field two will be treated as an alternate title. This means that the text string specified as a description in field two will be displayed on the screen at the defined cursor position. This description value will not be assigned a selection number, and thus can be used as a sub-title. This feature enables the MENU creator to define multiple titles and to have them displayed at any position on the screen. By using this feature, but leaving the title field (field one) null, a title can be displayed at the top of the screen, but without the current system time and date also being displayed.

Field four is multi-valued, and is associated value-by-value with the values in field two. Each value of field two is a single-line explanation of each selection on the MENU. The appropriate explanation is displayed when its associated selection number is typed, followed with a question mark, at the prompt for a selection to be made. For example, if "3?" is typed when the MENU processor prompts for a process selection, the explanation text for selection number three will be displayed (along with the command that would be executed if 3 were selected), and the selection prompt will be reissued.

Field five is single-valued, and can contain the user-defined cursor positioning of the query prompt and/or a user-defined query prompt. The query prompt is displayed whenever a MENU is displayed, and it prompts the user to choose one of the selections available on that MENU. If field five is null, the default value for the query prompt will be used. The default prompt is: "Which would you like? (1 - n)", where "n" is the highest selection number available on that particular MENU; this prompt is displayed two rows below the last selection number and description text displayed on the screen. If the MENU creator wishes to define a different query prompt, this is done by typing the desired prompt in field five when creating the MENU definition record. This would cause the specified query prompt to be displayed, at the default position. If the user desires to position the query prompt at some location other than the default position, then field five would be entered in one of the following formats:

```
(COL,ROW)

(COL,ROW)query.prompt.text
```

The first format will cause the default query prompt to be displayed at the postion defined by COL and ROW, while the second format will define "query.prompt.text" as the prompt text to be displayed at the postion defined by COL and ROW.

When defining the action field, one of the legal action values was specified to be the name of a MENU pointer record. This provides the abilty to nest MENUs (to invoke a new MENU by a selection from a different MENU). In other words, MENU "A" could invoke MENU "B", which in turn could invoke MENU "C", and so on to any reasonable level. Two fields are available in the MENU definition record to control the way in which these nested MENUs may be exited. They are field six (which contains legal "exit" commands), and field seven (which contains legal "stop" commands). The distinction between "exit" and "stop" is that an "exit" from a MENU will cause control to be passed to the process that invoked the MENU (which could be another MENU), while a "stop" will cause control to be passed all the way back to PERFORM's colon prompt regardless of whether the MENU processes have been nested or not.

Field six may contain a string of one or more user-defined "exit" commands, separated by commas. Each "exit" command is a string of text which has been selected by the creator of the MENU; typing any one of these strings of text in response to the MENU selection prompt will produce an exit from the MENU back to the processor that invoked it. (Note that if the person using the MENU does not know the contents of at least one of these "exit" text strings, it will not be possible for that user to do anything from that MENU except select one of its options; the MENU creator can take advantage of this feature to assure that a task using that MENU will be completely controlled by the MENU.)

If field six is null, then the "exit" command is the default "exit" command, which is <NEW LINE>.

Field seven may contain a list of legal "stop" commands, that are entered into the MENU definition record in the same format as the "exit" commands. These "stop" commands function differently from "exit" commands in that when one of them is entered at the query prompt, control is immediatly returned to PERFORM's colon prompt. If the user is in a MENU that is nested three levels deep, the entry of a "stop" command in response to the MENU's selection prompt will return control to PERFORM's colon prompt, while an "exit" command would return control to the prior MENU. Leaving field seven null will prohibit the user from returning to PERFORM's colon prompt from within a nested MENU; since there is no default value for the "stop" command, a null value in field seven results in having no mechanism for specifying "stop".

```
 MENU Definition Record

  id:  MENU name

 001:  MENU TITLE, or null
 002:  multi-valued list of process descriptions
 003:  multi-valued list of PERFORM commands
       (number of values should correspond to field two)
 004:  multi-valued list of process explanations
       (number of values should correspond to field two)
 005:  query prompt (to override default prompt)
 006:  list of valid "exit" commands, separated by comma's
       (or null, to use default <NEW LINE> "exit")
 007:  list of valid "stop" commands, separated by comma's
       (or null, to prohibit "stop")
```

FIGURE 1. Format of the MENU definition record.

```
  id:  MAIN.MENU

 001:
 002:    (5,0)Main MENU}(5,4)Inventory MENU}(5,6)Accounts Receivable
 003:  }INV.MENU}A/R.MENU
 004:  }Inventory Control MENU}Accounts Receivable MENU
 005:    (10,8)Enter Selection Number to Process
 006:  EXIT,EX,,QUIT,Q
 007:  STOP
```

FIGURE 2. Example of a MENU definition record.

4.6.2 MENUS: A MENU Maintenance Processor

A MENU maintenance processor which allows the user to create or modify a MENU, list the contents of a particular MENU, list all MENUs stored in a particular MENU file, or perform other MENU-related functions, is provided with the system; its name is MENUS.

A MENU maintenance processor which allows the user to create or modify a MENU, list the contents of a particular MENU, list all MENUs stored in a particular MENU file, or perform other MENU-related functions, is provided with the system. This processor, itself MENU-driven, is invoked by entering the command MENUS in response to the colon prompt from PERFORM. The format of the command is:

```
     MENUS
```

When this command is entered, the PRIME INFORMATION MENU Maintenance MENU is displayed on the user terminal, and an opportunity is given to select one of the operations. Selection 1 is used to create a standard MENU (one that does not use cursor positioning). Selection 2 is similar to selection 1, but allows for formatting the MENU, and additionally allows for the definition of "exit" and "stop" parameters. Selection 3 is used to list the names of the MENUs stored in a MENU file whose name is provided in response to a prompt. Selection 4 is used to display the contents of a particular MENU, after responding to a prompt for the name of the MENU, and the name of the MENU file where it is stored. Selection 5 is used to put a record into the VOC file by which a particular MENU can be invoked; this will be a type "M" record.

Selection 6 is used to create or modify a SENTENCE in the VOC file. Selection 7 is used to display all MENU selector items (records whose type is "M") in the VOC file. Selection 8 is used to display all stored SENTENCE records (records whose type is "S") in the VOC file. Selection 9 is used to display the contents of the DICTIONARY of a file whose name is provided in response to a prompt. Selection 10 is used to print (on the system printer) a summary of all MENUs on a MENU file whose name is provided in response to a prompt. Selection 11 is used to print (on the system printer) the contents of a MENU whose name, and the name of the MENU file where it is stored, are provided in response to prompts. Selection 12 is described in detail in the section titled "MENU.PIX Command" elsewhere in this manual. Selection 13 is used to print the contents of a formatted MENU on the system printer. Selection 14 is used to print the DICTIONARY of a file on the system printer. Selection 15 is described in detail in the section of this manual titled "MENU.DOC" command.

The normal sequence of operations used to create a new MENU are:

1. Create a new MENU file using the CREATE.FILE command described in this manual (or use an existing MENU file).

2. Define the MENU by using either Selection 1 or Selection 2 of the MENUS processor (depending on whether the MENU is to be formatted and/or use defined "exit" and "stop" commands).

3. Create the VOC file record by which the MENU will be invoked, using Selection 5 of the MENUS processor.

4. Optionally, document the MENU using MENU.PIX and/or MENU.DOC (which may be invoked by using Selection 12 or 13, or Selection 15, respectively, of the MENUS processor).

```
:MENUS
         Information MENU Maintenance      13:15:53 27 MAY 1980

 1= Enter/Modify   a MENU
 2= Enter/Modify   a formatted MENU
 3= Display        a summary of all MENUs on a MENU file
 4= Display        the contents of a MENU
 5= Enter/Modify   a VOC MENU selector item
 6= Enter/Modify   a VOC stored sentence item
 7= Display        all MENU selector items on the VOC file
 8= Display        all stored sentence items on the VOC file
 9= Display        the dictionary of a file
10= Print          a summary of all MENUs on a MENU file
11= Print          the contents of a MENU
12= Print          a virtual image of a MENU on the printer
13= Print          the contents of a formatted MENU
14= Print          the dictionary of a file
15= Print          detail of a MENU including VOC records referenced

Which would you like? ( 1 - 15 )=
```

FIGURE 1. The PRIME INFORMATION MENU Maintenance MENU.

4.7 REMOTE Item Definition Record

A REMOTE item is a record in the VOC file that points to a record in another file, where the remotely-referenced item is defined.

A REMOTE item is a record in the VOC file that points to a record in another file, where the remotely-referenced item is defined.

Remote item records in the VOC file contain the letter R as the first character in field one, to indicate that the record is a remote definition; field one may also contain a description. Field two contains the name of the file where the actual item is defined; this file name must be defined in the VOC file. Field three contains the record identifier of the actual item to be used by PERFORM in place of the remote item's name.

The use of Remote items for user-defined processes helps keep the VOC file small, by storing a brief remote-item pointer record in the heavily-used VOC file instead of the lengthy paragraphs or sentences it replaces. For example, the word END.OF.MONTH may be stored in the VOC file as a remote item, pointing to a long paragraph named EOM.PROCESS in a file called VOCLIB. The convenience of invoking this process by simply typing END.OF.MONTH in response to PERFORM's colon prompt is available, without having to store what is probably a very long paragraph in the central file of the PRIME INFORMATION system: the VOC file.

Remote items are also useful for defining sentences and paragraphs to be stored in different files according to the application in which they are used. In this way, all of the processes of a particular application may be kept together in a single file, for ease of maintenance.

The remote record pointed to has the same format as if it were in the VOC file, according to its type; e.g., if the remote item is a PARAGRAPH, then the record defining the item is identical to a PARAGRAPH record in the VOC file.

The stored sentence LISTR displays all remote definition records stored in the VOC file.

```
 id: remote name

001: R {description}
002: file.name
003: record.id
```

FIGURE 1. Format of the REMOTE item pointer record.

REMOTE item pointer in the VOC file:

```
 id: END.OF.MONTH

001: R Month-end processing
002: VOCLIB
003: EOM.PROCESS
```

Actual item in the VOCLIB file:

```
 id: EOM.PROCESS

001: PA End-of-month processing paragraph
002: SSELECT COMPANY.MASTER
003: EOM.ACCOUNTING
004: SSELECT CUSTOMER.FILE BY BILLING.DATE
005: EOM.BILLING
      .
      .
      .
      .
nnn: DISPLAY END-OF-MONTH PROCESSING COMPLETED
```

FIGURE 2. Example of the actual REMOTE item record.

4.8 VERB Definition Record

Verbs are defined by a record in the user's VOC file. The record directs the invocation of one of the INFORMATION processors.

The action taken by PERFORM for any VERB is determined by the VERB definition record in the VOC file. The VERB definition record has the following format:

```
 id: verb name

001: V {description}
002: processor name
003: dispatch type
004: processor mode
005 - nnn: reserved
```

The record must have a "V" as the first character of field one to define that this is a VERB definition record. Words two through the end of the field constitute the one line description of the VERB.

"Processor name" is the actual program that is executed.

The dispatch type codes are:

IB	cataloged INFO/BASIC programs	For user-written programs and PRIME INFORMATION implementation software
DI	Directives	Reserved for PRIME INFORMATION implementation software
IN	Informatives	Reserved for PRIME INFORMATION implementation software
PE	PERFORM internal	Reserved for PRIME INFORMATION implementation software
PR	PRIMOS commands	Reserved for PRIME INFORMATION implementation software

The "processor mode" is used by PERFORM or the INFORMATION processor named in field two. For example, if field four is "SELECT" or "SSELECT", PERFORM will not destroy a select list but will leave it intact for the next process which references the file.

It should be noted that all fields from five on are reserved.

The stored sentence LISTV displays all VERB definition records stored in the VOC file.

```
 id:  verb name

001:  V {description}
002:  processor name
003:  dispatch type (IB, DI, IN, PE, PR)
004:  processor mode
005:  reserved
   .
   .
   .
nnn:  reserved
```

FIGURE 1. Format of the VERB Definition Record.

ASSIGN	AVAIL	BASIC	BLOCK.PRINT
BLOCK.TERM	BTD	CATALOG	CD
CHAP	CLEAN.ACCOUNT	CLEAR.FILE	CLEAR.LOCKS
CLEARDATA	CLEARSELECT	CLR	CNAME
COMO	COMPILE.DICT	COPY	COUNT
CREATE.FILE	CS	DATE	DATE.FORMAT
DELAY	DELETE	DELETE.CATALOG	DELETE.FILE
DELETE.LIST	DISPLAY	DTB	DTO
DTX	ED	ENTER	ENTRO
ENTROC	FILE.STAT	FORM.LIST	FORMAT
GET.LIST	GO	GROUP.STAT	GROUP.STAT.DETAIL
HASH.HELP	HASH.HELP.DETAIL	HASH.TEST	HASH.TEST.DETAIL
HELP	HUSH	IAM	IF
ISTAT	LIST	LIST.LOCKS	LIST.READU
LO	LOCK	LOGOUT	LOGTO
LOOP	MAIL	MAKE.MAP.FILE	MAP
MASTER	MENU.DOC	MENU.PIX	MESSAGE
OFF	OTD	P.ATT	P.DET
PASSWD	PHANTOM	PROP	PSPOOL
PTERM	PTIME	QUIT	RADIX
RECORD	RELEASE	RELEASE.ALL	RELEASE.ITEMS
REPEAT	RESET.PRINTER	RESIZE	RUN
SAVE.LIST	SELECT	SETFILE	SETPTR
SLEEP	SORT	SPOOL	SSELECT
STAT	STATUS	SUM	T.ATT
T.BCK	T.DET	T.DUMP	T.FWD
T.LOAD	T.READ	T.REW	T.WEOF
TERM	TIME	UNASSIGN	USERS
VCATALOG	WHO	XTD	

FIGURE 2. Verbs supplied with the PRIME INFORMATION System.

4.9 FILE Definition Record

Each PRIME INFORMATION System file is defined by a record in the VOC file.

When a PRIME INFORMATION file is created, the CREATE.FILE processor adds a record to the VOC file which defines the file being created. The format of this record is:

```
 id:  file.name

001:  F {description}
002:  DATA file name
003:  DICTIONARY file name
```

The default values used when the file is created are:

```
DATA file name       = file.name
DICTIONARY file name = D_file.name
```

If it is desired to have a synonym name by which the file could be referenced (in addition to referencing it by "file.name"), all that is needed is to copy the "file.name" record in the VOC file to a "new.name" record in the VOC file. (See also the section titled "SETFILE Command" in this manual.)

Since the DATA and DICT sections of a file are maintained by PRIME INFORMATION as physically separate files, any file may be described by several different dictionaries. This may be done by changing field three (DICTIONARY file name) of a synonym file definition record to the name of the dictionary to use when the file is referenced by that synonym. Similarly, one dictionary may be used for many data files. The CLEAR.FILE and DELETE.FILE commands should be used with care if these features are in use. (See the sections titled "CLEAR.FILE Command" and "DELETE.FILE Command" in this manual.)

A file definition record in the VOC file may also point to files in other accounts, through the use of PRIMOS treenames in fields two and three. The SETFILE command is useful in establishing these remote file pointer records. (See the section titled "SETFILE Command" in this manual.)

The stored sentence LISTF displays all file definition records in the VOC file. The stored sentence LISTFL displays all file definition records in the VOC file which are <u>local</u> files (i.e., do not use PRIMOS treenames to point to files which are stored in other accounts). The stored sentence LISTFR displays all file definition records in the VOC file which are <u>remote</u> files (i.e., use PRIMOS treenames to point to files which are stored in other accounts).

```
 id:       CUSTOMER.MASTER

001:       F Customer Master File
002:       CUST.MAST
003:       D_CUST.MAST
```

FIGURE 1. Example of a typical FILE definition record.

```
STANDARD CUSTOMER MASTER          SPECIAL CUSTOMER MASTER
 id:  CUSTOMER.MASTER              id:  CUSTOMER.MASTER.X

001:  F                           001:  F
002:  CUST.MAST                   002:  CUST.MAST
003:  D_CUST.MAST                 003:  D_CUST.MAST.RESTRICTED
```

FIGURE 2. Example of a file with more than one dictionary.

```
INVENTORY MASTER                  PARTS MASTER
 id:  INVENTORY.MASTER             id:  PARTS.MASTER.X

001:  F                           001:  F
002:  INV.MASTER                  002:  PARTS.MASTER
003:  D_INV.MASTER                003:  D_INV.MASTER
```

FIGURE 3. Example of two files with a single dictionary.

```
 id:  AP.CUST

001:  F
002:  ACCAP>CUST
003:  ACCAP>D_CUST
```

"AP.CUST" is a synonym file name for a file in account "ACCAP". The DATA section of the file is "CUST", identified by the PRIMOS treename "ACCAP>CUST", and the DICT section of the file is "D_CUST", identified by the PRIMOS treename "ACCAP>D_CUST".

FIGURE 4. Example of a file in another account.

4.10 KEYWORD Definition Record

Keywords are used by PRIME INFORMATION System verbs to define special operations.

Keywords are used by PRIME INFORMATION to define special operations, such as MATCHING, WITH, BY, etc. When PERFORM encounters a keyword in a sentence, it searches the VOC file for a record named "keyword", and substitutes a coded operation number for the keyword.

The format of the keyword record in the VOC file is:

```
 id:  keyword.name

001:  K {description}
002:  keyword.opcode
```

where "keyword.opcode" is a number which defines the operation for "keyword.name".

It is possible to create synonyms for existing keywords, by creating new records in the VOC file which use the same format shown above, with the appropriate operation number in field two of the record. For example, a new keyword ORDERING could be defined by copying the keyword record for BY. It would then be possible to type

```
:LIST INVENTORY ORDERING DATE.RECEIVED
```

and get the same results as from

```
:LIST INVENTORY BY DATE.RECEIVED
```

The stored sentence LISTK displays all of the keywords defined in the VOC file.

```
id:  keyword.name

001: K {description}
002: keyword.opcode
```

FIGURE 1. Format of the KEYWORD record.

```
id:  NO.PAGE            id:  BREAK.ON        id:  OVERWRITING

001: K                  001: K               001: K
002: 30                 002: 16              002: 34
```

FIGURE 2. Examples of keyword records.

>	3	AFTER	3	GREATER	3	GT	3
=	4	EQ	4	EQUAL	4	=>	5
>=	5	GE	5	<	6	BEFORE	6
LESS	6	LT	6	#	7	NE	7
NOT	7	<=	8	=<	8	LE	8
OR	9	&	10	AND	10	LIKE	11
MATCHES	11	MATCHING	11	BY	12	BY-DSND	13
BY.DSND	13	BUGS	14	TOTAL	15	BREAK-ON	16
BREAK.ON	16	DET-SUPP	17	DET.SUP	17	ID-SUP	18
ID.SUP	18	HEADER	19	HEADING	19	DICT	20
DBL-SPC	21	DBL.SPC	21	FIRST	22	SAMPLE	22
FOOTING	23	TEMPL	24	USING	24	NOT.MATCHING	25
UNLIKE	25	ID.ONLY	26	ONLY	26	WITH	27
HDR-SUPP	28	HDR.SUP	28	COL-SUPP	29	COL.SUP	29
NO.PAGE	30	NOPAGE	30	SAMPLED	31	COL.SPACES	32
COL.SPCS	32	(P)	33	LPTR	33	OVERWRITING	34
DELETING	35	SQUAWK	36	ALL	37	A	38
ANY	38	ARE	38	FILE	38	FOR	38
IN	38	INVISIBLE	38	OF	38	PRINT	38
THAN	38	THE	38	CALC	39	CALCULATE	39
MTU	40	BLK	41	BREAK.SUP	42	%	44
PCT	44	PERCENT	44	PERCENTAGE	44	SAID	45
SPOKEN	45	~	45	MARGIN	46	AVERAGE	47
AVG	47	BY.EXP	48	BY.EXP.DSND	49	VERT	50
VERTICALLY	50	EVERY	51	VERIFILE	52	VERIFY	52
WHEN	53	VERIFIELD	54	TEMPLATE	55	REQUIRE.SELECT	56
NEXT.AVAILABLE	57	INQUIRING	58				

FIGURE 3. List of KEYWORDS in opcode sequence.

4.11 USER Record Definition

A reserved user record type X has been provided, to allow anyone to put into the VOC file records which will not be treated as invalid records by PERFORM.

Every record in the VOC file has a type code in field one of the record. This type code is used by PERFORM to determine the type of each record, and PERFORM must be able to control the allocation and meaning of these type codes. The type codes presently defined are listed in the section titled "The User Vocabulary File", elsewhere in this manual. All type codes which are not listed are reserved for possible future use.

Some users wish to place items in the VOC file which are to be used in daily processing, for whatever purpose the user may define. In order to provide a guarantee of non-interference with future type codes which may be defined by PERFORM, a special user-defined record, with type code X, has been provided. PERFORM will always ignore type X records (with the two exceptions listed below), so that this type code may be used for these user-defined purposes. No other type code should be "invented" and used in the VOC file: the definition of type codes is reserved by PRIME.

The format of the user record in the VOC file is:

```
    id: record.id

    001: X {description}
```

The user may use fields two, three, etc. for whatever data is meaningful to the user-defined purpose of the record.

Presently, PRIME INFORMATION uses two type X records. The first of these is used to verify that the user's VOCABULARY has been updated to agree with the cataloged version of PRIME INFORMATION which is currently executing. This type X record is named RELLEVEL, and contains in field three the release level of the VOC file at the time it was created or last updated. This VOC file release level is verified against an internal release level kept within PERFORM, and if the two do not agree, PERFORM produces a warning message.

The other type X record used by PRIME INFORMATION in the VOC file is the record named "STACKWRITE". This record is used by PERFORM's sentence stack processor, to determine whether to maintain the user's sentence stack in a file so that it is preserved after LOGOFF. If field two of this record contains "OFF", the sentence stack preservation routine is "turned off", and the stack is not maintained after LOGOFF.

The stored sentence LISTO produces a list of all records in the VOC file which either are not defined in the section of this manual titled "VOC File Record Types" (there should be none of these), or which are type X or type I.

```
 id: record.id

001: X {description}
```

Fields two, three, etc. can be used for anything.

FIGURE 1. Format of the USER record.

```
 id: RELLEVEL

ØØ1: X Item used by PERFORM to define RELEASE LEVEL of account
ØØ2: 4.Ø
```

FIGURE 2. The PRIME INFORMATION user record RELLEVEL.

```
 id: STACKWRITE

ØØ1: X If field two = "OFF", stack is not saved after LOGOFF.
ØØ2: ON
```

FIGURE 3. The PRIME INFORMATION user record STACKWRITE.

4.12 Stored System Sentences

The VOC file contains a miscellaneous group of stored sentences that may be used to list the contents of the VOC file belonging to a single record type.

The stored sentences are:

LISTF	lists all files defined in your VOC file.
LISTFL	lists local files defined in your VOC file.
LISTFR	lists remote files defined in your VOC file.
LISTK	lists keywords defined in your VOC file.
LISTM	lists menus defined in your VOC file.
LISTPA	lists paragraphs defined in your VOC file.
LISTPH	lists phrases defined in your VOC file.
LISTPR	lists PRIMOS commands defined in your VOC file.
LISTR	lists remotes defined in your VOC file.
LISTS	lists sentences defined in your VOC file.
LISTV	lists verbs defined in your VOC file.
LISTO	list all other things not listed above that are defined in your VOC file.

Note that any of these sentences may be used with "LPTR" appended to it, to direct the listing to the system printer instead of to your terminal.

EXAMPLE:

```
:LISTV LPTR
```

```
:LISTF NO.PAGE

These are the FILES defined in your VOCABULARY
NAME............. FILE................ DICTIONARY.....

&PH&               &PH&
&SAVEDLISTS&       &SAVEDLISTS&         D_&SAVEDLISTS&
&UFD&              *                    D_&UFD&
APP.PROGS          APP.PROGS            D_APP.PROGS
DICT.DICT          ISYS>DICT.DICT       ISYS>DICT.DICT
ENTRO-PROCESSES    ISYS>ENTRO.PROCESSES ISYS>D_ENTRO.PROCESSES
ENTRO.DISCUSSIONS  ENTRO.DISCUSSIONS    D_ENTRO.DISCUSSIONS
ENTRO.PROCESSES    ISYS>ENTRO.PROCESSES ISYS>D_ENTRO.PROCESSES
ISYS.VOCLIB        ISYS>VOCLIB          ISYS>D_VOCLIB
KWM                ISYS>KWM             ISYS>D_KWM
MENU.FILE          MENU.FILE            D_MENU.FILE
NEWACC             ISYS>NEWACC          ISYS>D_NEWACC
VERBS              VERBS                D_VERBS
VOC                VOC                  D_VOC
VOCLIB             VOCLIB               D_VOCLIB
15 RECORDS LISTED
```

FIGURE 1. Example of use of a stored system sentence.

---

5 PRIME INFORMATION COMMANDS

5.1 * (COMMENT) Statement

The "*" statement is provided to allow insertion of comments into stored system paragraphs. These may be used to help document the intent of the paragraph.

The "*" statement is provided to allow insertion of comments into stored system paragraphs. The format of the * (COMMENT) statement is:

```
     * comment.text
```

When any sentence is encountered by PERFORM which begins with an asterisk (*), that sentence is ignored. It is not printed on the terminal, nor is it examined for "correctness"; however, the comment is examined to see if it contains a prompting sequence, and if there is one, the prompting is done. (See the section titled "Prompting Within a Stored PERFORM Command" in this manual.)

You may use a comment statement not only in a paragraph (its normal use), but also from the terminal. This would be particularly useful if you were using a COMO file, and wished to document an event for further examination later. (See the section titled "COMO Command" in this manual).

For a form of the COMMENT statment which additionally prints "comment.text" on the terminal, see the section titled "DISPLAY Command" in this manual.

```
     * comment.text
```

FIGURE 1. Format of the * (COMMENT) Statement.

With this Paragraph stored in the VOC file:

```
      id: INV.RUN.1

     001: PA
     002: * First, list the Inventory file with QOH less than 100.
     003: INV.LIST
     004: DATA 100
     005: * Now, generate orders for items in short supply.
     006: RUN INV INV.SHORT
     007: * Finally, print journal of items ordered.
     008: RUN INV INV.JOURNAL
```

The following would be the result at the terminal:

```
     :INV.RUN.1
     :
```

FIGURE 2. Example of the * (COMMENT) Statement.

5.2 ASSIGN Command

The ASSIGN command may be issued at any user terminal, to request exclusive control of a peripheral device.

A user can request exclusive control of a peripheral device with the ASSIGN command. The format of this command is:

```
ASSIGN device {-WAIT}
```

where "device" is an available peripheral. The most common peripherals are:

MTn	Magnetic tape units, "n" from 0 to 7
CRn	Card reader, "n" from 0 to 3
PTR	Paper tape reader
PUNCH	Paper tape punch

If "device" is currently assigned to another user, the system replies:

```
DEVICE IN USE
```

unless the optional argument "-WAIT" was supplied. In this case, the ASSIGN command is queued until "device" is UNASSIGNed by the other user, or until the other user logs out. (See the section titled "UNASSIGN Command" in this manual.)

If any I/O operation is attempted which requires exclusive control of a peripheral device, and that device has not been assigned to the task with the ASSIGN command, the error message:

```
DEVICE NOT ASSIGNED
```

is displayed at the terminal.

For additional details on the ASSIGN command and on peripheral devices that can be ASSIGNed, see the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108.

```
    ASSIGN device {-WAIT}
```

FIGURE 1. Format of the ASSIGN command.

:<u>ASSIGN MT1</u>	assign magnetic tape unit 1
:<u>ASSIGN CRØ</u>	assign card reader unit Ø
:<u>ASSIGN MT1 -WAIT</u>	assign magnetic tape unit 1, wait for it if necessary

FIGURE 2. Examples of the ASSIGN command.

5.3 AVAIL Command

The AVAIL command displays disk usage statistics at the terminal.

The AVAIL command displays disk usage statistics at the terminal. These statistics include the number of normalized disk records used, the number of normalized disk records still available, and the percent of total logical disk space already used. The format of the command is:

```
      AVAIL {argument}
```

where the values of "argument" (if specified) are:

packname	The MFD name of the logical disk whose statistics are wanted.
-LDEV n	The logical disk whose number is "n".
*	Use the file named DISCS in UFD SYSTEM as a list of logical devices whose statistics are wanted.
number	The spelled-out number of the logical disk whose statistics are wanted. (i.e., "number" must be ONE, TWO, THREE, etc.)

If no "argument" is specified, AVAIL displays the disk usage statistics of the logical disk where the user's account is located.

Note that the number of records displayed is "normalized" to a physical record size of 880 bytes per disk sector. This normalization is done to maintain compatability with early users of PRIME computers. Today's disks have sector sizes of 2048 bytes. To determine disk space usage in bytes, multiply number of records times 880. To determine disk space usage in sectors, multiply number of records times 880 and divide the result by 2048.

For additional explanations of these options and the meaning of the displayed data, see the section titled "Monitoring the System" in THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109. Also see the section on the AVAIL command in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108.

```
     AVAIL {argument}
```

FIGURE 1. Format of the AVAIL command.

```
:AVAIL *
VOLUME      TOTAL       FREE    %
  ID        RECS        RECS   FULL     COMMENTS
----------------------------------------------------
INF4.0      68952       8350   87.9     INFORMATION RELEASE 4.0
MASTER NOT MOUNTED
SOURCE NOT MOUNTED
DEMO        68952       6290   90.9     DEMO ACCOUNTS
BENCH       34476        895   97.4     BENCHMARK ACCOUNTS

:AVAIL ONE
VOLUME DEMO
68952 TOTAL RECORDS (NORMALIZED)
6290 RECORDS AVAILABLE (NORMALIZED)
90.9% FULL

:AVAIL DEMO
VOLUME DEMO
68952 TOTAL RECORDS (NORMALIZED)
6290 RECORDS AVAILABLE (NORMALIZED)
90.9% FULL

:AVAIL
VOLUME INF4.0
68952 TOTAL RECORDS (NORMALIZED)
8350 RECORDS AVAILABLE (NORMALIZED)
87.9% FULL
```

FIGURE 2. Examples of the AVAIL command.

5.4 BASIC Command

The BASIC command is used to compile INFO/BASIC source code for execution. This command uses a SELECT list if one is active.

To compile an INFO/BASIC source program, the PRIME INFORMATION user types the following command:

```
     BASIC file.name {program.name} ...  {-option} ...
```

The "program.name" parameter may be a list of program names to be compiled, or it may be omitted as a signal to the INFO/BASIC compiler to use a SELECT list. If a SELECT list is used, the INFO/BASIC compiler will ignore all records beginning with "$" and with "x_" (where "x" is any letter). These records are ignored because, by convention, records which begin with "$" are compiled object code, and those which begin with "x_" are $INSERT files.

INFO/BASIC program source code must be stored in a PRIME INFORMATION type 1 file (a file type which corresponds to a PRIMOS sub-ufd). Programs may be created and edited with either the PRIME INFORMATION Editor, or with the standard PRIME Editor. (For further information, see THE INFO/BASIC REFERENCE GUIDE, PDR3903.)

The "-option" is an optional control specification to the compiler (with the minus sign a required element of the option name). The following options are presently defined:

-LIST or -L	produce a listing of the compilation, named "L_program.name" in "file.name"
-XREF or -X	produce a cross-reference listing of labels and variables
-SPOOL or -S	produce a listing of the compilation, and spool it direct to the line printer
-OPTA	compile with Microdata REALITY DATA/BASIC compatiblity

The "-LIST" option generates a listing of the program, while "-XREF" generates a cross-reference table of all labels and symbolic variable names used in the program. The "-LIST" and "-XREF" listings are saved in "file.name" under the name "L_program.name" (i.e., the "program.name" prefixed by "L_"). The "-SPOOL" option is the same as the "-LIST" option, except that the listing is spooled directly to the printer instead of being filed. The "-OPTA" option causes compilation to be performed with compatibility to Microdata REALITY DATA/BASIC. (See THE INFO/BASIC REFERENCE GUIDE, PDR3903.)

When the "-LIST" option is used, the listing will show line numbers, all lines inserted with $INSERT (which do not increment the source program line numbers), and all symbolic string substitutions specified by any EQUATE statements.

The object code produced by the compiler will be stored in "file.name" under the name "$program.name" (i.e., the "program.name" prefixed by "$"). During compilation, an asterisk ("*") is displayed for each 10 lines of code compiled.

```
    BASIC file.name {program.name} ... {-option} ...
```

The options are:

-LIST or -L	generate a listing record named "L_program.name" in "file.name"
-XREF or -X	generate a cross-reference table of all labels and symbolic variable names used in the program
-SPOOL or -S	generate a listing and spool it to the line printer
-OPTA	causes compilation to be performed with compatibility to Microdata REALITY DATA/BASIC.

FIGURE 1. Format of the BASIC command.

```
:BASIC BP.AP RPT.PGM1 -SPOOL -XREF
```

This command will compile the source code of the program named "RPT.PGM1" in the file named "BP.AP", and will cause a listing and a cross-reference to be spooled to the line printer. The compiled object code will be stored in the source file "BP.AP", under the name "$RPT.PGM1".

FIGURE 2. Example of the BASIC command.

5.5 BLOCK.PRINT Command

The BLOCK.PRINT command is used to print characters in block format on the system printer.

The BLOCK.PRINT command will print on the system printer the block format equivalent of the character string supplied with the command. As an example of block format, the letter A in block format would be several lines comprised of the letter A alternating with spaces, to form a large character which looks like the letter A. The format of the BLOCK.PRINT command is:

```
     BLOCK.PRINT character.string ...
```

Each individual character of "character.string" is printed in a 9 by 'n' block. The variable 'n' ranges from 5 to 20, depending on the relative width of the character to be printed (the letter "I" is the narrowest character, and the letters "M" and "W" are the widest). The number of characters which may be contained in the character string is limited by the output width of the character string. The total output width of the string must be less than or equal to the line width available on the printer. If this limit is exceeded, an error message will be displayed, indicating that the character string is too long to be printed in block character format. Multiple "character.string"s (separated by blanks) will be printed on the system printer on separate lines.

The BLOCK.PRINT command uses the file named BLTRS, which must be defined in the VOC file of your account. The BLTRS file exists in the ISYS account, and is used to store a "map" of each character that can be printed in block format. Any character that does not exist in the BLTRS file cannot be printed in block format by the BLOCK.PRINT command.

See also the description of the BLOCK.TERM command, which is similar to the BLOCK.PRINT command, except that the block format equivalent of "character.string" is displayed on the terminal.

```
BLOCK.PRINT character.string ...
```

FIGURE 1. Format of the BLOCK.PRINT Command.

```
:BLOCK.PRINT HELLO THERE
```

* The following is what would be printed on the system printer:

```
HHHH    HHHH   EEEEEEEEEEE    LLLL            LLLL              OOOOOOOOOO
HHHH    HHHH   EEEEEEEEEEE    LLLL            LLLL             OOOOOOOOOOOO
HHHH    HHHH   EEEE           LLLL            LLLL             OOOO    OOOO
HHHHHHHHHHHH   EEEEEEEEE      LLLL            LLLL             OOOO    OOOO
HHHHHHHHHHHH   EEEEEEEEE      LLLL            LLLL             OOOO    OOOO
HHHH    HHHH   EEEE           LLLL            LLLL             OOOO    OOOO
HHHH    HHHH   EEEE           LLLL            LLLL             OOOO    OOOO
HHHH    HHHH   EEEEEEEEEEE    LLLLLLLLLLL     LLLLLLLLLLL      OOOOOOOOOOOO
HHHH    HHHH   EEEEEEEEEEE    LLLLLLLLLLL     LLLLLLLLLLL       OOOOOOOOOO

TTTTTTTTTTTT   HHHH    HHHH   EEEEEEEEEEE   RRRRRRRRRRR    EEEEEEEEEEE
TTTTTTTTTTTT   HHHH    HHHH   EEEEEEEEEEE   RRRRRRRRRRRR   EEEEEEEEEEE
    TTTT       HHHH    HHHH   EEEE          RRRR    RRRR   EEEE
    TTTT       HHHHHHHHHHHH   EEEEEEEEE     RRRR    RRRR   EEEEEEEEE
    TTTT       HHHHHHHHHHHH   EEEEEEEEE     RRRRRRRRRRRR   EEEEEEEEE
    TTTT       HHHH    HHHH   EEEE          RRRRRRRRRRR    EEEE
    TTTT       HHHH    HHHH   EEEE          RRRR  RRRR     EEEE
    TTTT       HHHH    HHHH   EEEEEEEEEEE   RRRR   RRRR    EEEEEEEEEEE
    TTTT       HHHH    HHHH   EEEEEEEEEEE   RRRR    RRRR   EEEEEEEEEEE
```

FIGURE 2. Example of the BLOCK.PRINT command.

5.6 BLOCK.TERM Command

The BLOCK.TERM command is used to print characters in block format on the terminal.

The BLOCK.TERM command will print on the terminal the block format equivalent of the character string supplied with the command. As an example of block format, the letter A in block format would be several lines comprised of the letter A alternating with spaces, to form a large character which looks like the letter A. The format of the BLOCK.TERM command is:

```
     BLOCK.TERM character.string ...
```

Each individual character of "character.string" is printed in a 9 by 'n' block. The variable 'n' ranges from 5 to 20, depending on the relative width of the character to be printed (the letter "I" is the narrowest character, and the letters "M" and "W" are the widest). The number of characters which may be contained in the character string is limited by the output width of the character string. The total output width of the string must be less than or equal to the line width available on the terminal. If this limit is exceeded, an error message will be displayed, indicating that the character string is too long to be printed in block character format. For most 80-column terminals, the effective maximum character string size is five characters. Multiple "character.string"s (separated by blanks) will be printed on the terminal on separate lines.

The BLOCK.TERM command uses the file named BLTRS, which must be defined in the VOC file of your account. The BLTRS file exists in the ISYS account, and is used to store a "map" of each character that can be printed in block format. Any character that does not exist in the BLTRS file cannot be printed in block format by the BLOCK.TERM command.

See also the description of the BLOCK.TERM command, which is similar to the BLOCK.PRINT command, except that the block format equivalent of "character.string" is printed on the system printer.

```
BLOCK.TERM character.string ...
```

FIGURE 1. Format of the BLOCK.TERM Command.

```
:BLOCK.TERM HELLO THERE

HHHH    HHHH  EEEEEEEEEEE   LLLL          LLLL           OOOOOOOOOO
HHHH    HHHH  EEEEEEEEEEE   LLLL          LLLL          OOOOOOOOOOOO
HHHH    HHHH  EEEE          LLLL          LLLL          OOOO    OOOO
HHHHHHHHHHHH  EEEEEEEEE     LLLL          LLLL          OOOO    OOOO
HHHHHHHHHHHH  EEEEEEEEE     LLLL          LLLL          OOOO    OOOO
HHHH    HHHH  EEEE          LLLL          LLLL          OOOO    OOOO
HHHH    HHHH  EEEE          LLLL          LLLL          OOOO    OOOO
HHHH    HHHH  EEEEEEEEEEE   LLLLLLLLLLL   LLLLLLLLLLL   OOOOOOOOOOOO
HHHH    HHHH  EEEEEEEEEEE   LLLLLLLLLLL   LLLLLLLLLLL    OOOOOOOOOO

TTTTTTTTTTTT  HHHH    HHHH  EEEEEEEEEEE   RRRRRRRRRRR   EEEEEEEEEEE
TTTTTTTTTTTT  HHHH    HHHH  EEEEEEEEEEE   RRRRRRRRRRRR  EEEEEEEEEEE
    TTTT      HHHH    HHHH  EEEE          RRRR    RRRR  EEEE
    TTTT      HHHHHHHHHHHH  EEEEEEEEE     RRRR    RRRR  EEEEEEEEE
    TTTT      HHHHHHHHHHHH  EEEEEEEEE     RRRRRRRRRRRR  EEEEEEEEE
    TTTT      HHHH    HHHH  EEEE          RRRRRRRRRRR   EEEE
    TTTT      HHHH    HHHH  EEEE          RRRR  RRRR    EEEE
    TTTT      HHHH    HHHH  EEEEEEEEEEE   RRRR   RRRR   EEEEEEEEEEE
    TTTT      HHHH    HHHH  EEEEEEEEEEE   RRRR    RRRR  EEEEEEEEEEE
```

FIGURE 2. Example of the BLOCK.TERM command.

5.7 CATALOG Command

The CATALOG command is used to to copy into system catalog space the object code generated by the compilation of an INFO/BASIC program.

After compilation, object code from an INFO/BASIC program may be copied into system catalog space by the CATALOG verb. Catalog space is accessible to all people on the system, and allows having only one copy of the program in the system, regardless of how many people are executing the program. If two or more people are running the same INFO/BASIC program, there is a copy of that program in system memory for each user, unless it has been cataloged; this puts a much larger demand on real memory and may affect response time. In general, programs that are going to be used by more than one user should be cataloged. External subroutines that are to be called from a program must be cataloged, although the main (calling) program need not be.

The format of the CATALOG command is:

```
     CATALOG {file.name} {catalog.name} {program.name} {FORCE}
             {NOXREF} {LOCAL}
```

where "catalog.name" is the name under which the program will be stored in the catalog space; in the case of an external subroutine, this must be the name used in the CALL statement of the main program. The parameter "file.name" is the name of the file where the object code is stored, and "program.name" is the name of the program whose object code is to be cataloged. If only "file.name" is entered, the CATALOG processor will prompt for "catalog.name" and "program.name". If only "file.name" and "program.name" are entered, then "catalog.name" and "program.name" will be the same. The "catalog.name" frequently will be the same as "program.name", but with a special prefix. Programs cataloged with "catalog.name" prefixed with "*" or "-" are classified as globally cataloged programs; all others are classified as locally cataloged programs. The prefix character "*" is provided for user-written programs; the prefix "-" is reserved for PRIME INFORMATION verbs.

The optional "FORCE" keyword specifies that, if the program being cataloged already exists in system catalog space as a GLOBALLY available program, it is to be overwritten; if this parameter is not used, the CATALOG command will prompt for verification that an existing program may be overwritten.

The optional "NOXREF" keyword specifies that no cross reference data or symbol table is to be stored with the program; note that this will handicap PERFORM's debugging assistance during execution of the program, and so should not be used except for programs that have been thoroughly tested.

The optional "LOCAL" keyword specifies that the program is to be treated as if it were cataloged, but is not to be moved to system catalog space; only the account from which it was cataloged will be able to access this program.

Locally cataloged programs require a verb definition record in the VOC file, which is created automatically when a program is cataloged; see Figure 2 for an example. Users on other accounts do not have access to a locally cataloged program unless the local program definition record is copied to the VOC file in the other account.

Globally cataloged programs are retrieved more rapidly than are locally cataloged programs, because PERFORM does not have to access the VOC file to find the program in the catalog space. If the main program is cataloged, the VOC file of every account that is to be able to use the program must contain a VERB definition record for "program.name".

```
CATALOG {file.name} {catalog.name} {program.name} {FORCE}
        {NOXREF} {LOCAL}
```

The CATALOG processor will prompt for parameters not entered at the same time as the command is typed.

FIGURE 1. Format of the CATALOG Command.

If a user is in account CURLEY and enters the following:

```
:CATALOG
catalog name =MAIL.LABELS
file name =ADMIN.SOURCE
program name =MAIL.LABELS
```

The following record will be put in the VOC file:

```
 id: MAIL.LABELS
001: V
002: *CURLEY*MAIL.LABELS
003: IB
```

This program could also be cataloged by the one-line command:

```
:CATALOG ADMIN.SOURCE MAIL.LABELS
```

This program is a locally cataloged program.

FIGURE 2. Example of the CATALOG Command.

5.8 CD Command

Before using dictionary Information ("I" type) descriptors in an INFORM sentence, they must be compiled. Compilation is accomplished by using the CD command, which is an abbreviation of the COMPILE.DICT command.

Before an "I" descriptor can be referenced in an INFORM sentence, it must be compiled by the COMPILE.DICT command. An abbreviated form of the command, CD, is provided for convenience. For a complete description of the CD command, see the section titled "COMPILE.DICT Command" elsewhere in this manual.

5.9 CHAP Command

Each user has limited control over his or her execution priority through the use of the CHAP command.

The PRIME INFORMATION System has four levels of execution priority, ranging from 0 (lowest execution priority) to 3 (highest execution priority). The system default execution priority for all users is 1.

A user at a user terminal can change his or her priority (between 0 and the defined upper limit) by using the following commands:

```
CHAP DOWN

CHAP

CHAP UP
```

CHAP DOWN will lower the user's priority by 1 each time it is used, down to the lowest priority limit of 0. CHAP UP will raise the user's priority by 1 each time it is used, up to the defined upper limit for that task (which is normally 1). CHAP with no parameters is the same as CHAP DOWN. A user priority upper limit can be set to 2 or 3 only at the supervisor terminal. Note that unless the system administrator has changed a task's upper execution priority, CHAP DOWN will normally result in setting the task's execution priority to 0, and CHAP UP will normally result in setting the task's execution priority to 1.

The STATUS command will display a user's priority, if it is other than the default. (See the section titled "STATUS Command" in this manual.)

For additional details concerning the CHAP command and its interaction with the PRIMOS-level CHAP command, see the CHAP command in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, and the CHAP command in the THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109.

CHAP DOWN	lower the user execution priority by 1 (normally, set execution priority to 0)
CHAP UP	raise the user execution priority by 1 (normally, set execution priority to 1)

FIGURE 1. Format of the CHAP command.

```
:WHO
2 KOVSKY
:STATUS US

 USER                                  NO LIN PDEVS
KOVSKY                                  2  0  1060
SIMONDS                                 4  2  41060
SYSTEM                                 12 77  1060 PR0
:CHAP DOWN
:STATUS US

 USER                                  NO LIN PDEVS
KOVSKY                                  2  0  1060 (0)
SIMONDS                                 4  2  41060
SYSTEM                                 12 77  1060 PR0

:CHAP UP
:STATUS US

 USER                                  NO LIN PDEVS
KOVSKY                                  2  0  1060
SIMONDS                                 4  2  41060
SYSTEM                                 12 77  1060 PR0
```

FIGURE 2. Examples of the CHAP command.

5.10 CLEAN.ACCOUNT Command

The CLEAN.ACCOUNT command is provided for the System Administrator to use as part of routine system maintenance.

The CLEAN.ACCOUNT command is used by the System Administrator for routine maintenance of your account, or sometimes to correct suspected problems with the files in your account. The format of the command is:

```
          CLEAN.ACCOUNT
```

The CLEAN.ACCOUNT command performs a number of file-related functions. If the CLEAN.ACCOUNT process finds a file named "&TEMP&" in your account, it will ask if that file may be deleted; if the reply is affirmative, this temporary file will be deleted. If there is a file named "&PH&" in your account (created and used by the PHANTOM command, described in this manual), CLEAN.ACCOUNT will clear this file (but not delete it) if you indicate that is what you wish done.

CLEAN.ACCOUNT also searches the file named "&SAVEDLISTS&" (if there is one for the account) for all records whose name begins with "S#" or with "&". These records are temporary records, used respectively for SORT work files and as unnamed savelists. If any are found, CLEAN.ACCOUNT deletes them. All other records in the "&SAVEDLISTS&" file are left intact.

Next, CLEAN.ACCOUNT verifies every item in your VOC file that is type "F" (for file), checking that the physical file exists in your UFD, that the file type and modulus are correct, etc. Any errors or inconsistencies are reported via messages on your terminal, with a suggested corrective action.

```
    CLEAN.ACCOUNT
```

FIGURE 1. Format of the CLEAN.ACCOUNT command.

```
:CLEAN.ACCOUNT
OK to delete &TEMP&? (YES or NO) =YES
Deleting file &TEMP&, Type 1.
OK to clear &PH&? (YES or NO) =YES
&PH& has been cleared.
EXECUTING "SSELECT VOC WITH TYPE = F "

27 records selected.

EXECUTING "SSELECT &UFD&"

26 records selected.

***********************
The following were left as unresolved mysteries in this account
A PRIMOS file in your account called O_CLEAN.ACCOUNT
```

FIGURE 2. Example of the CLEAN.ACCOUNT command.

5.11 CLEAR.FILE Command

The data or dictionary sections of a file may be cleared of all records by using the CLEAR.FILE command. This command uses a SELECT list if one is active.

The CLEAR.FILE command is used to clear a file (either the data or dictionary section, or both) of all records. The format of the command is:

```
          CLEAR.FILE {DATA} {file.name}

                    or

          CLEAR.FILE {DICT} {file.name}
```

If neither the DATA nor the DICT keyword is entered at the same time as the command, both the DATA and the DICT portions of the file will be cleared. As with other PERFORM commands, if the command is entered with no parameters at all, it will prompt for "file.name".

The process is break-inhibited; that is, once the actual clearing process has begun, the process cannot be interrupted by the BREAK key.

The file to be cleared can be either an INFO/DMS direct access data file or dictionary (file type 2 through 9), or a PRIME INFORMATION type 1 file (sub-ufd). "File.name" may be prefaced with either the word "DATA" or "DICT" to clear only the appropriate section.

This command may be used with an active SELECT list, if desired. The SELECT list would be comprised of names of files (record identifiers for records in the VOC file with type "F" in field one). After activating the SELECT list (see the SELECT command in THE INFORM REFERENCE GUIDE, IDR3905, or the GET.LIST and FORM.LIST commands in this manual), the CLEAR.LIST command is invoked without a "file.name" parameter. In this mode, the CLEAR.LIST command processor will verify that you intended to use the active SELECT list by prompting you for an affirmative response.

```
     CLEAR.FILE {DATA} {file.name}

                 or

     CLEAR.FILE {DICT} {file.name}
```

FIGURE 1. Format of the CLEAR.FILE command.

```
:CLEAR.FILE TRANSACTIONS
TRANSACTIONS has been cleared
```

FIGURE 2. Example of the CLEAR.FILE command.

5.12 CLEAR.LOCKS Command

The CLEAR.LOCKS command releases either a specified task synchronization lock, or all task synchronization locks, for the user invoking the command.

PRIME INFORMATION systems have 64 user-accessible semaphores which are referred to as execution locks. These locks may be used to synchronize processes running on different terminals on the same system. The meaning of any one of these locks is defined by your System Administrator. For example, Lock Number 23 may be defined (within your installation) as being the lock that must be set when any process which could affect a General Ledger balance is being executed. If all programs which modify Accounting balances correctly set Lock Number 23 before they actually modify these balances, then the integrity of the General Ledger balance is assured.

Any specified execution lock set by your task may be cleared (i.e., set to the -1 status) by the CLEAR.LOCKS command. The format of the command is:

```
     CLEAR.LOCKS {lock.number}
```

where "lock.number" is the particular lock to clear. If "lock.number" is omitted then all 64 execution locks set by your task are cleared.

WARNING: It is your responsiblity to understand the use of these task synchronization locks by your installation, before you use this command. If you use the CLEAR.LOCKS command to release a lock before it should have been released, you may affect the integrity of the information being maintained on your system. The CLEAR.LOCKS command will clear the specified "lock.number" (or all locks if no "lock.number" is specified) without regard to the dependence of any currently-executing programs on the lock(s) being left in the state set by those programs; the only requirement is that the lock was set by your task. If you are not absolutely certain of the effects of this command, you should have your System Administrator accomplish your intended purpose by whatever mechanism he or she thinks is appropriate.

See also the description of the LIST.LOCKS and LOCK commands elsewhere in this manual. The System Administrator may also wish to review the MASTER command, described in this manual.

```
     CLEAR.LOCKS {lock.number}
```

where "lock.number" is the lock to be cleared. If "lock.number" is omitted then all locks that were set by your task are cleared.

FIGURE 1. Format of the CLEAR.LOCKS command.

```
:LOCK 7
:LOCK 9
:LOCK 11
:LIST.LOCKS

 Ø: -1 !  1: -1 !  2: -1 !  3: -1 !  4: -1 !  5: -1 !  6: -1 !  7:  Ø !
 8: -1 !  9:  Ø ! 1Ø: -1 ! 11:  Ø ! 12: -1 ! 13: -1 ! 14: -1 ! 15: -1 !
16: -1 ! 17: -1 ! 18: -1 ! 19: -1 ! 2Ø: -1 ! 21: -1 ! 22: -1 ! 23: -1 !
24: -1 ! 25: -1 ! 26: -1 ! 27: -1 ! 28: -1 ! 29: -1 ! 3Ø: -1 ! 31: -1 !
32: -1 ! 33: -1 ! 34: -1 ! 35: -1 ! 36: -1 ! 37: -1 ! 38: -1 ! 39: -1 !
4Ø: -1 ! 41: -1 ! 42: -1 ! 43: -1 ! 44: -1 ! 45: -1 ! 46: -1 ! 47: -1 !
48: -1 ! 49: -1 ! 5Ø: -1 ! 51: -1 ! 52: -1 ! 53: -1 ! 54: -1 ! 55: -1 !
56: -1 ! 57: -1 ! 58: -1 ! 59: -1 ! 6Ø: -1 ! 61: -1 ! 62: -1 ! 63: -1 !

:CLEAR.LOCKS 7
:LIST.LOCKS

 Ø: -1 !  1: -1 !  2: -1 !  3: -1 !  4: -1 !  5: -1 !  6: -1 !  7: -1 !
 8: -1 !  9:  Ø ! 1Ø: -1 ! 11:  Ø ! 12: -1 ! 13: -1 ! 14: -1 ! 15: -1 !
16: -1 ! 17: -1 ! 18: -1 ! 19: -1 ! 2Ø: -1 ! 21: -1 ! 22: -1 ! 23: -1 !
24: -1 ! 25: -1 ! 26: -1 ! 27: -1 ! 28: -1 ! 29: -1 ! 3Ø: -1 ! 31: -1 !
32: -1 ! 33: -1 ! 34: -1 ! 35: -1 ! 36: -1 ! 37: -1 ! 38: -1 ! 39: -1 !
4Ø: -1 ! 41: -1 ! 42: -1 ! 43: -1 ! 44: -1 ! 45: -1 ! 46: -1 ! 47: -1 !
48: -1 ! 49: -1 ! 5Ø: -1 ! 51: -1 ! 52: -1 ! 53: -1 ! 54: -1 ! 55: -1 !
56: -1 ! 57: -1 ! 58: -1 ! 59: -1 ! 6Ø: -1 ! 61: -1 ! 62: -1 ! 63: -1 !

:CLEAR.LOCKS
:LIST.LOCKS

 Ø: -1 !  1: -1 !  2: -1 !  3: -1 !  4: -1 !  5: -1 !  6: -1 !  7: -1 !
 8: -1 !  9: -1 ! 1Ø: -1 ! 11: -1 ! 12: -1 ! 13: -1 ! 14: -1 ! 15: -1 !
16: -1 ! 17: -1 ! 18: -1 ! 19: -1 ! 2Ø: -1 ! 21: -1 ! 22: -1 ! 23: -1 !
24: -1 ! 25: -1 ! 26: -1 ! 27: -1 ! 28: -1 ! 29: -1 ! 3Ø: -1 ! 31: -1 !
32: -1 ! 33: -1 ! 34: -1 ! 35: -1 ! 36: -1 ! 37: -1 ! 38: -1 ! 39: -1 !
4Ø: -1 ! 41: -1 ! 42: -1 ! 43: -1 ! 44: -1 ! 45: -1 ! 46: -1 ! 47: -1 !
48: -1 ! 49: -1 ! 5Ø: -1 ! 51: -1 ! 52: -1 ! 53: -1 ! 54: -1 ! 55: -1 !
56: -1 ! 57: -1 ! 58: -1 ! 59: -1 ! 6Ø: -1 ! 61: -1 ! 62: -1 ! 63: -1 !
```

FIGURE 2. Examples of the CLEAR.LOCKS command.

5.13 CLEARDATA Command

The CLEARDATA command allows an INFO/BASIC program to clear out the data stack that has been built either from DATA statements in a paragraph, or from DATA statements in another INFO/BASIC program.

Within PRIME INFORMATION, it is possible to build a data stack of pre-supplied responses to anticipated INPUT statements that will be executed by an INFO/BASIC program. This data stack can be built either by a series of DATA statements in a paragraph, or by a series of DATA statements in an INFO/BASIC program. If such a data stack has been built, and an INFO/BASIC program executes an INPUT statement, the next unused element of the data stack will be supplied to the program as if it had been typed at the terminal. When there are no more elements left on the data stack (or if there was no data stack created), any subsequent INPUT statement executed by the INFO/BASIC program will require a response from the terminal.

Occasionally, it may be necessary for an INFO/BASIC program to "throw away" the rest of the elements of a pre-supplied data stack, and to obtain its input from the terminal. As an example, if a program was invoked from a paragraph that contained DATA statements to build a data stack, and the program encountered an error in its processing (such as a missing file, with the file name having been supplied as a DATA statement in the data stack), that program might choose to abandon the remaining elements of the data stack, and request a new (valid) file name from the terminal operator. The CLEARDATA command is supplied for this purpose.

The format of the CLEARDATA command is:

```
     EXECUTE 'CLEARDATA'
```

Note that, although this command appears in an account's VOC file, it is not intended for use at the terminal. It is useful only when it is invoked via an EXECUTE statement within an INFO/BASIC program. Note also that all unused elements of a data stack are erased by PERFORM each time your task returns to the colon (:) prompt; therefore, it is not necessary for an INFO/BASIC program to use the CLEARDATA command except in unusual situations.

```
EXECUTE 'CLEARDATA'
```

FIGURE 1. Format of the CLEARDATA command.

```
GET.FILE.NAME:
      PRINT "Enter name of file ":
      INPUT FILE.NAME
      OPEN "", FILE.NAME TO FILE.VAR ELSE
         EXECUTE "CLEARDATA"
         PRINT "Unable to open file ":FILE.NAME
         GOTO GET.FILE.NAME
      END
      PRINT "Enter name of record to be modified ":
      INPUT RECORD.NAME
    . . .
```

In the program fragment of this example, the program will prompt for file name, and later for record name. If the file name is not correct and the program re-prompts for it, the next DATA statement (if the program was invoked with a DATA statement stack) will be provided as the response; this is obviously invalid. By executing the CLEARDATA command, the program can assure that the data being input matches its intended use.

FIGURE 2. Example of the CLEARDATA command.

5.14 CLEARSELECT Command

Active SELECT lists that have been constructed by the SELECT or SSELECT processors, or by the FORM.LIST or GET.LIST commands, may be cancelled before they would have been used by subsequent processes. The PERFORM verb which cancels an active SELECT list is CLEARSELECT.

A SELECT list is made active by executing a SELECT or SSELECT command, by executing a GET.LIST command, or by executing a FORM.LIST command. If the command which immediately follows the activation of a SELECT list is one which utilizes a SELECT list, then it will use the list just activated. If you have activated such a SELECT list, and then decide that you do not wish to have the next command or program use it after all, you may use the CLEARSELECT verb to cancel the SELECT list.

The format of the CLEARSELECT command is:

```
     CLEARSELECT
```

Many PRIME INFORMATION processors use a SELECT list if one is present at the time that they are invoked. For example, most of the INFORM commands such as LIST, COUNT, SUM, SELECT, etc., use an active SELECT list if there is one, as do most of the PERFORM commands which are file-related (such as GROUP.STAT, DELETE, ED, etc.). Additionally, any INFO/BASIC program which utilizes the READNEXT statement will use a SELECT list that had already been created at the time the program was invoked. Note that, in the case of an INFO/BASIC program using the READNEXT statement, an active SELECT list may be partially processed and then the remainder disposed of, with the CLEARSELECT command (which is also an INFO/BASIC statement).

```
CLEARSELECT
```

FIGURE 1. Format of the CLEARSELECT Command.

```
:SELECT EMPLOYEES WITH DEPENDENTS > 3
49 items selected.
:COUNT EMPLOYEES
49 records counted.

:SELECT EMPLOYEES WITH DEPENDENTS > 3
49 items selected.
:CLEARSELECT
:COUNT EMPLOYEES
328 records counted.
```

FIGURE 2. Example of the CLEARSELECT Command.

5.15 CLR Command

The CLR command is provided to allow the user to erase the terminal display. The CLR command is a synonym for the CS command.

It may be desirable to erase the data last displayed on the terminal, as, for example, when sensitive information has just been displayed but is no longer needed. To prevent casual observers from seeing this sensitive information, the display can be erased with the CLR command.

The format of the CLR command is as follows:

```
     CLR
```

This command clears the terminal screen.

Note that the CLR command relies on the ability of the terminal to recognize and respond to a particular control character sequence. Since there is no standard among terminal manufacturers for the control sequences which generate various responses (such as CLEAR SCREEN), the CLR command will not work on every terminal. It is specifically intended to work on PRIME's PT25 terminal.

While this command may work on terminals which do not use the control sequences of PRIME's PT25 terminals, it is unlikely. In this case, the user may write a simple INFO/BASIC program which writes the appropriate control sequence to erase the terminal display, and replace the VOC record for CLR with a pointer to this cataloged INFO/BASIC program.

```
PRINT CHAR(12):STR(CHAR(0),20):@(0,0)
END
```

FIGURE 1. Actual statement executed to CLR screen.

```
PRINT CHAR(27):"K"
END
```

FIGURE 2. Example of user-written CLR (for Perkin-Elmer FOX terminal).

5.16 CNAME Command

The CNAME command is used to change the name of a PRIME INFORMATION file, or to change the names of records within a PRIME INFORMATION file.

The CNAME command is used to change the name of a PRIME INFORMATION file, or to change the names of records within a file. The format of the command to change file names is:

```
     CNAME old.file.name,new.file.name

                  or

     CNAME old.file.name TO new.file.name
```

Either format of the CNAME command will rename "old.file.name" to "new.file.name". CNAME will make the necessary changes to the PRIME INFORMATION file index "/FILE.DEF/", and also to the VOC file. If the VOC file entry whose record identifier is "old.file.name" is a synonym (that is, the DATA file name field and/or the DICT file name field are not the same as "old.file.name"), CNAME will affect only the appropriate field of the VOC file entry (see FIGURE 3 for an example).

```
     CNAME  file.name  old.rec.id,new.rec.id  ...

                  or

     CNAME file.name old.rec.id TO new.rec.id
```

Either of these formats of the CNAME command will rename "old.rec.id" to "new.rec.id" in the file "file.name". The first form of this command allows specification of a list of records to be renamed within a file by continuing the list of old and new record names (separated by commas) on the same command line. The second form of this command allows only a single pair of old and new record names per command line.

```
    CNAME  old.file.name,new.file.name

                   or

    CNAME  old.file.name  TO  new.file.name


    CNAME  file.name  old.rec.id,new.rec.id  ...

                   or

    CNAME  file.name  old.rec.id  TO  new.rec.id
```

FIGURE 1. Formats of the CNAME command.

```
:CNAME BP TO TYPE1

:CNAME SALES.ORDERS S123,A123 S124,A124

:CNAME INVOICES,PAID.INVOICES
```

FIGURE 2. Example of the CNAME command.

```
VOC file item "SALES.ORDERS"
     001: F
     002: SALES.ORDERS
     003: D_INVOICES

:CNAME SALES.ORDERS,SALES.HISTORY

VOC file item "SALES.HISTORY"
     001: F
     002: SALES.HISTORY
     003: D_INVOICES
```

FIGURE 3. Example of the CNAME command for a VOC file synonym item.

5.17 COMO Command

The COMO command allows the user to manipulate PRIMOS COMO files from within PRIME INFORMATION.

The format of the command is:

```
    COMO {action} {file.name} {param}
```

where "action" may be:

ON	- create a new COMOUTPUT file
OFF	- stop capturing terminal display
SPOOL	- print a COMOUTPUT file to lineprinter
DELETE	- delete COMOUTPUT file
LIST	- list all COMOUTPUT files for an account

"file.name" is the name of the COMOUTPUT file to be accessed; and "param" may be:

HUSH	- a keyword to indicate that the terminal display should be captured on the COMOUTPUT file, but not displayed on the terminal. HUSH is used only with the ON action.
T	- to direct a SPOOL list to the terminal.

If the COMO verb is entered without any options, the command processor will prompt for the necessary parameters.

The COMOUTPUT files created by the COMO command are automatically prefixed with "O_", to identify them as COMOUTPUT files at the PRIMOS level. Within PRIME INFORMATION, the user should refer to these files by the "file.name" specified (omitting the "O_").

A note of caution on the use of the HUSH keyword: if this keyword is used with an INFORM LIST statement, carriage returns are still required for each page of output, unless the INFORM statement contains the NO.PAGE parameter. The HUSH keyword is equivalent to the -NTTY keyword of the PRIMOS COMO command. See the description of the HUSH command, elsewhere in this manual, for additional information about the effects of this option.

The COMO command is a PRIME INFORMATION-level version of the PRIMOS COMO command, described in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108. Note that the files produced by this command can become quite large, and are not automatically deleted by the system. This is a powerful and useful tool, but should not be used indiscriminately. When old COMO files are no longer needed, they should be deleted.

```
     COMO {action} {file.name} {param}
```

FIGURE 1. Format of the COMO command.

```
:COMO

 Enter action (ON,OFF,DELETE,LIST,SPOOL,QUIT) =ON
 COMO file name or QUIT  =TUESDAY
 Terminal display on?  (Y/N)  =Y
TUESDAY established


 Enter action (ON,OFF,DELETE,LIST,SPOOL,QUIT) =LIST


COMO file list              15:18:47 12 MAR 1980
   1 POSTING
   2 LOCK
   3 GEN.LEDGER
   4 ACCT.REC
   5 ACCT.PAY
   6 ORD.ENT
   7 MAIL.LIST
   8 TUESDAY          IN USE

Enter number of file to be selected or QUIT=3

Enter action (ON,OFF,DELETE,LIST,SPOOL,QUIT) =DELETE
GEN.LEDGER deleted


Enter action (ON,OFF,DELETE,LIST,SPOOL,QUIT) =QUIT
End of COMO session


:COMO SPOOL ORD.ENT
ORD.ENT spooled to line printer
```

FIGURE 2. Examples of the COMO command.

5.18 COMPILE.DICT Command

Before using dictionary Information ("I" type) descriptors in an INFORM sentence, they must be compiled. Compilation is accomplished by using the COMPILE.DICT command.

Before an "I" descriptor can be referenced in an INFORM sentence, it must be compiled. (For a description of "I" descriptors, see the section titled "INFORMATION DESCRIPTORS" in THE INFORM REFERENCE GUIDE, IDR3905). The COMPILE.DICT command can specify a particular descriptor to be compiled, or it can compile the entire dictionary. A short form of the verb, COMPILE.DICT, is provided for convenience. The possible formats of the COMPILE.DICT or COMPILE.DICT command are:

```
    COMPILE.DICT file.name {descriptor.name}...

                  or

    COMPILE.DICT file.name {descriptor.name}...
```

If an optional "descriptor.name" is not specified, all the "I" descriptors in the dictionary are compiled.

The compiled object code for the descriptor expression is stored in the descriptor record, along with the time and date of the compilation.

Whenever one or more "I" descriptors are changed in a dictionary, it is advisable to compile <u>all</u> the "I" descriptors in that dictionary, since other "I" descriptors may have referenced those that have been changed.

When the PRIME INFORMATION EDITOR is used to change "I" descriptors, it invalidates the object code stored from any previous compilation, as a signal to INFORM that this item requires compilation before use. At the time that the EDITOR is asked to FILE the changed "I" descriptor, it will issue a reminder message that the item needs to be compiled. Because of the "invalidated" object code now stored with the "I" descriptor, INFORM will notice that the item has not been compiled if that item is used in an INFORM sentence, and will automatically compile it before the sentence is executed.

Note that, at present, ENTRO does <u>not</u> invalidate the object code of previously-compiled "I" descriptors which it modifies; the responsibility for compiling these ENTRO-modified "I" descriptors is the user's.

```
COMPILE.DICT filename {descriptor.name}...

        or

COMPILE.DICT filename {descriptor.name}...
```

FIGURE 1. Format of the COMPILE.DICT Command.

```
 id:  field.name

ØØ1:  I {desc}     type code and optional description
ØØ2:  expression   expression specification
ØØ3:  conversion   conversion specification
ØØ4:  name         optional display name
ØØ5:  format       format specification
ØØ6:  s/m          single-valued or multi-valued
ØØ7:  assoc        association name
ØØ8 thru Ø15       <reserved>
Ø16:  object       compiled object code
Ø17:  time         time (internal format) last compiled
Ø18:  date         date (internal format) last compiled
Ø19:  object       compiled object code
Ø2Ø and on         <reserved>
```

FIGURE 2. An "I" descriptor record after compilation.

5.19 COPY Command

The COPY command is used to copy records from one file to another, or to copy records within a file and give them new record identifiers. The COPY command uses a SELECT list if one is active.

The COPY command is used to move records from one file to another, optionally deleting the source file records, changing the identifiers of the source file records in the target file, and overwriting existing records in the target file which have the same identifiers as the source records. Both the source file and the target file may be either DATA portions of PRIME INFORMATION files, or DICT portions of these files. The COPY command may also be used within a single file, to copy records while giving them new identifiers, optionally deleting the original records and optionally overwriting existing records which have the same new identifier. The format of the COPY command is as follows:

```
    COPY FROM {DICT} file1 {TO {DICT} file2} {rec.list}
         {OVERWRITING} {DELETING} {SQUAWK}
```

"rec.list" can be one of three forms:

```
    rec1 rec2 rec3

          or

    rec1,rec2   rec3,rec4

          or

    ALL
```

If the first form of "rec.list" is used, the named records will be copied with no change in their record identifiers. If the second form of "rec.list" is used, records "rec1" and "rec3" will be copied and their record identifiers changed to "rec2" and "rec4", respectively. In form three of "rec.list", the keyword "ALL" directs the COPY processor to copy all records from the "FROM" file. If a select list is used, "rec.list" is not specified. (See the description of the SELECT and SSELECT verbs in THE INFORM REFERENCE GUIDE, IDR3905).

If the source record identifier already exists on the target file, it will not be overwritten unless the "OVERWRITING" keyword is included. If you are copying to a newly-created file, or to a newly-cleared file (making an archival backup, for example), the performance of COPY can be improved greatly by using the OVERWRITING keyword, since the COPY process does not have to read each record in the target file before writing it. Source records are not deleted by COPY, unless the "DELETING" keyword is included.

If the "SQUAWK" option is specified, the copy processor outputs a descriptive narrative as the records are being copied. (See example in Figure 2.) If the "TO" file.name is not specified, the COPY process operates within the "FROM" file that was specified.

Note that the COPY command should not be used to copy compiled object code (i.e., "$program.name" records, where the "$" indicates object code). Note also that copying dictionaries which contain compiled "I" items should always be followed by the command ":COMPILE.DICT file.name", since the "I" items in a dictionary are often inter-related and should always be compiled together. Finally, note that the COPY command may not be used with mixed file types: i.e., to copy from File Type 1 to File Type 2 through 9, or vice versa.

```
COPY FROM {DICT} file1 {TO {DICT} file2} {rec.list}
     {OVERWRITING} {DELETING} {SQUAWK}
```

FIGURE 1. Format of the COPY command.

```
:COPY FROM CANDIDATE TO MEMBER ALL SQUAWK

* from file =  CANDIDATE
* to file   =  MEMBER
* orig.file.sw = 1
* dest.file.sw = 1
* overwriting.sw = 0
* deleting.sw    = 0
JONES copied to JONES
SMITH copied to SMITH
BROWN copied to BROWN
DOAKES copied to DOAKES
HOWARD copied to HOWARD
5 records copied.
```

FIGURE 2. Example of the COPY command with the SQUAWK option.

5.20 COUNT (INFORM System Processor)

COUNT is a portion of the high-level, natural language QUERY and REPORT GENERATOR portion of the PRIME INFORMATION system. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. COUNT is one of these INFORM processors.

5.21 CREATE.FILE Command

The CREATE.FILE command is used to create files to be used with the PRIME INFORMATION system.

All PRIME INFORMATION files must be created by the CREATE.FILE command. The format of the CREATE.FILE command is:

```
     CREATE.FILE {dict/data} {file.name} {type} {modulo}  {description}
```

CREATE.FILE normally creates both data and dictionary sections for the file, but the optional "dict/data" parameter may be used to specify that only the DICT portion of the file, or only the DATA portion of the file, is to be created, by using the keyword DICT or DATA as the "dict/data" parameter. CREATE.FILE also creates the file definition record in the VOC file, by which the file being created will be referenced. The data portion of the file is named "file.name", and the dictionary portion is named "D_file.name"; these files appear as PRIMOS files in the account's ufd. (See the section titled "FILE Definition Records" in this manual.)

Two categories of files may be created: hash-encoded direct access, or sub-ufd. Hash-encoded direct access files should be used for all data files; sub-ufd files should be used for program source or RUNOFF source. There are eight different file types within the "hash-encoded direct access" category; these are files types 2 through 9. The sub-ufd file is specified by choosing file type 1. When the CREATE.FILE processor prompts for file type, a brief description of the primary purpose of each one is displayed in response to a "?" entry. (See the section titled "PRIME INFORMATION Files" elsewhere in this manual for a complete description of each file type.)

"Modulo" is used for direct access data files only, and must be specified as an integer with values between 1 and 32,767, inclusive.

The optional parameter "description" is added to the file definition record in the VOC file, as the comment portion of field one (after the "F" which identifies the VOC file record as a file definition record). This "description" will be displayed in response to the sequence ":.? file.name".

The CREATE.FILE processor will prompt for input if the command is entered with no parameters. (See example in Figure 2.)

```
    CREATE.FILE {dict/data}  {file.name} {type} {modulo} {description}
```

The CREATE.FILE processor will prompt for input. Type "?" at the "ENTER FILE TYPE" prompt to have the list of file types displayed. In the multi-line form of the CREATE.FILE command, the CREATE.FILE processor will prompt for an optional "description" in addition to prompting for "file.name", "type" and "modulo". This file description becomes the remainder of field one in the VOC file record defining the file (after the "F" which identifies the record as a File definition record), and is the data to be displayed in response to a user typing "?file.name" at the PERFORM colon prompt.

FIGURE 1. Format of the CREATE.FILE command.

```
:CREATE.FILE INV
File type        =?

The valid file types are:

Type  Usage
====  =====
  1   Sub-ufd type file, for large records or source code
  2   Hashed, keys end in numbers
  3   Hashed, keys end mainly in numbers
  4   Hashed, keys end in letters
  5   Hashed, keys end in full range of ASCII characters
  6   Hashed, keys begin in numbers
  7   Hashed, keys begin mainly in numbers
  8   Hashed, keys begin in letters
  9   Hashed, keys begin in full range of ASCII characters

File type        =2
Modulo           =7
File description =Master INVENTORY file.
Creating file INV / Type 2 / Modulo 7.
Creating file D_INV (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_INV.
```

The same file may be created with the following one-line command format:

```
:CREATE.FILE INV 2 7 Master INVENTORY File
```

FIGURE 2. Example of CREATE.FILE command.

5.22 CS Command

The CS command is provided to allow the user to erase the terminal display.

It may be desirable to erase the data last displayed on the terminal, as, for example, when sensitive information has just been displayed but is no longer needed. To prevent casual observers from seeing this sensitive information, the display can be erased with the CS command.

The format of the CS command is as follows:

```
     CS
```

This command clears the terminal screen.

Note that the CS command relies on the ability of the terminal to recognize and respond to a particular control character sequence. Since there is no standard among terminal manufacturers for the control sequences which generate various responses (such as CLEAR SCREEN), the CS command will not work on every terminal. It is specifically intended to work on PRIME's PT25 terminals.

While this command may work on terminals which do not use the control sequences of PRIME's PT25 terminals, it is unlikely. In this case, the user may write a simple INFO/BASIC program which writes the appropriate control sequence to erase the terminal display, and replace the VOC record for CS with a pointer to this cataloged INFO/BASIC program.

```
      PRINT CHAR(12):STR(CHAR(0),20):@(0,0)
      END
```

FIGURE 1. Actual statement executed to CS screen.

```
      PRINT CHAR(27):"K"
      END
```

FIGURE 2. Example of user-written CS (for Perkin-Elmer FOX terminal).

5.23 DATE Command

The DATE command displays the current system date and time at the user terminal.

The current system date and time are printed at the user terminal by the DATE command. The DATE command requires no parameters; its format is simply:

```
     DATE
```

The date and time displayed by invoking the DATE verb are the date and time set by the System Administrator at the system operator's console. This system date is a required parameter of the PRIMOS bootstrap process, but may be reset at any time after that process. The DATE command is provided as a simple mechanism for verifying that the correct system date and time have been set, before beginning a process that requires the correct date and time.

The format of the date portion of the display is the standard format used in the United States. For example:

```
     Friday, February 29, 1980
```

This format may be changed to European format by executing the DATE.FORMAT verb; see the description of the DATE.FORMAT verb in this manual.

To obtain the system date in a different format, which includes the time of day, see the "TIME Command" section elsewhere in this manual.

```
    DATE
```

FIGURE 1. Format of the DATE command.

```
:DATE
Thursday, March 20, 1980  02:14 PM
```

FIGURE 2. Example of the DATE command (standard format).

```
:DATE
Thursday, 20 March 1980  02:14 PM
```

FIGURE 3. Example of the DATE command (European format).

5.24 DATE.FORMAT Command

The DATE.FORMAT verb changes the default format for dates from American format to European format.

The default format for dates on the PRIME INFORMATION system is the American format, which is month followed by day followed by year. This format can be modified to European format (day followed by month followed by year) by the DATE.FORMAT command. The format of this command is:

```
DATE.FORMAT
```

The DATE.FORMAT verb has no effect on the internal representation of dates (as described in the section titled "DATE() FUNCTION" in THE INFO/BASIC REFERENCE GUIDE, PDR3903), but only on the results of conversion from that internal representation to external representation. Specifically, the DATE.FORMAT verb will alter the date format produced by the following:

HEADING and FOOTING
DATE command
ICONV and OCONV

If the conversion field of a D descriptor in a dictionary were "D2/", and the contents of that field were the internal representation for the date September 25, 1980, then this field would normally be printed as:

```
09/25/80
```

However, if the DATE.FORMAT verb were used, this same date would now be printed as:

```
25/09/80
```

Note that the same result can be achieved by using "D2/E" in the D descriptor in the dictionary and not using the DATE.FORMAT command. Note also that a "D2/E" conversion specification will effectively become a "D2/" conversion specification, after execution of the DATE.FORMAT verb.

DATE.FORMAT is a "one-shot" command, rather than a "toggle" command. That is, once it has been executed, a second execution of the same command will not reset the date format to the default. If it is desired to return to the default date format, it is necessary to leave the PRIME INFORMATION environment (see the section titled "QUIT Command" in this manual) and then re-enter it (see the section titled "INFO Command" in this manual).

DATE.FORMAT

FIGURE 1. Format of the DATE.FORMAT command.

CONVERSION	BEFORE DATE.FORMAT	AFTER DATE.FORMAT
OCONV(4000,"D2/")	12/13/78	13/12/78
OCONV(4000,"D2/E")	13/12/78	12/13/78
ICONV("12/13/78","D2")	4000	null; STATUS() = 1
ICONV("12/13/78","D2E")	null; STATUS() = 1	4000

FIGURE 2. Results of the DATE.FORMAT command.

5.25 DELAY Command

The DELAY verb defines a time function to be used to delay the printing of a character after a Carriage Return or Carriage Return/Line Feed, to support hard copy terminals.

Since CRT terminals operate at electronic speeds, there is no need to delay data transmission after a Carriage Return or Carriage Return/Line Feed sequence; the cursor will have moved to the beginning of the next line before the first character of that line arrives. However, most hard-copy or printing terminals incur considerable mechanical delays in the physical movement of the printing mechanism back to the beginning of the line. With these types of terminals, failure to delay next-character transmission can cause the first few character(s) after a Carriage Return to be printed incorrectly.

The DELAY verb defines a time function to be used to delay the printing of the next character after a Carriage Return (<NEW LINE>) has been sent to a terminal. The format of the DELAY command is:

```
     DELAY {octal.minimum.delay} {octal.maximum.delay} {octal.r.margin}
```

where "octal.minimum.delay" defines the number of character-times (the time it takes the system to type a character on a line) to delay when <NEW LINE> is output at the left margin; "octal.maximum.delay" defines the number of character-times to delay when <NEW LINE> is output at the defined right-most position of the line; and "octal.r.margin" defines the right-most position of the line (expressed as the number of characters per line). If <NEW LINE> is output at some point within a line, the time delay is proportional to the number of characters typed. If "octal.r.margin" is not specified, 72 is assumed; if "octal.maximum.delay" is not specified, 12 is assumed. If the verb DELAY is given with no parameters, the default values 6, 12, and 72 are assumed. These values are adequate for most 30 cps terminals.

The DELAY process is actually implemented by sending ASCII "NULL" characters after each carriage return, depending on where the carriage (or cursor) was located at the time of the carriage return. Systems which have only CRT terminals might benefit from a slight performance improvement by using the following command in a LOGIN procedure:

```
     DELAY 1 1 120
```

This command specifies that only one delay character (ASCII NULL) will be transmitted after a Carriage Return, regardless of where the cursor was positioned. (Note that it is not possible to specify zero delay characters, since this command treats a parameter value of zero as a specification to return to the default value.)

The DELAY verb can be used prior to logging in, at the PRIMOS level. (See the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, for more information on this command.)

```
     DELAY {octal.minimum.delay} {octal.maximum.delay} {octal.r.margin}
```

FIGURE 1. Format of the DELAY command.

```
     :DELAY 0 10 100
```

will set the following delays:

0	Will use the default 6 character-time delays when a carriage return is encountered at the left margin. (0 specifies to use default parameter value.)
10	Decimal 8 character-time delays when a carriage return is encountered at the right margin.
100	The maximum expected line length is decimal 64 characters.

FIGURE 2. Example of the DELAY command.

5.26 DELETE Command

The DELETE command deletes a specified record or records from a named data file or dictionary. This command uses a SELECT list if one is active.

The DELETE command is used to delete records from the DATA section or the DICT section of a PRIME INFORMATION file. The format of this command is:

```
     DELETE {DICT} file.name record.id {record.id} . . .
```

Note that the "file.name" parameter, and at least one "record.id", must be specified on the command line with the DELETE command. Additional "record.id"'s may be specified, if desired.

If the DICT keyword is used, the DELETE processor will attempt to delete the specified "record.id"(s) from the dictionary portion of the named file "file.name". If the DICT keyword is omitted, the DELETE processor will attempt to delete the specified "record.id"(s) from the data portion of the named file.

The DELETE command also works with a SELECT LIST. After creating the select list (with the SELECT/SSELECT command), or after re-activating a saved select list (with the GET.LIST command), or after activating a user-created select list (with the FORM.LIST command), simply type "DELETE file.name", with no parameters (or type "DELETE DICT file.name" with no additional parameters). The DELETE processor will display the key of the first record in the select list, and ask for verification of your intent to delete all the records whose keys are in the select list. If you affirm that intention, DELETE will use the select list; otherwise, you will see the message "DELETE aborted" and will return to PERFORM's colon prompt. If you forget to activate a SELECT list, and type "DELETE file.name" or "DELETE DICT file.name", you will see the message "Not found. 0 records deleted.", and will return to PERFORM's colon prompt.

```
DELETE {DICT} file.name record.id {record.id} . . .
```

FIGURE 1. Format of the DELETE Command.

```
:DELETE PAYROLL SMITH JONES
2 records deleted.


:DELETE DICT CUSTOMERS PAST.DUE PHONE F29
3 records deleted.


:DELETE INVENTORY "321*10029*A4*7"
1 records deleted.
```

FIGURE 2. Examples of the DELETE Command.

```
:SELECT VENDORS WITH LAST.ORDER.DATE BEFORE 12/31/77
17 records selected.
:DELETE VENDORS
Do you wish to delete the records previously selected?
The first record ID=Acme.Indust.Supply
Enter Y or N
Y

17 records deleted.
```

FIGURE 3. Example of the DELETE command with a SELECT list.

5.27 DELETE.ACCOUNT Command

The DELETE.ACCOUNT command is provided for the exclusive use of the System Administrator, to delete PRIME INFORMATION accounts which are no longer needed on the system.

The DELETE.ACCOUNT command is used by the System Administrator for deleting PRIME INFORMATION accounts which are no longer needed on the system. Within a PRIMOS UFD that has been a PRIME INFORMATION account, the DELETE.ACCOUNT processor will delete only PRIME INFORMATION-related files; any files within the PRIMOS UFD which were created outside the PRIME INFORMATION environment will be left intact. Additionally, the DELETE.ACCOUNT processor will delete from the system CATALOG space any programs which were cataloged from within the account being deleted. The format of the command is:

```
     DELETE.ACCOUNT
```

Because of the potentially disastrous results of its uninformed use, the DELETE.ACCOUNT command is protected from inadvertant use. Its operation is documented here, but its correct invocation is left to your System Administrator. Once invoked, the DELETE.ACCOUNT processor will prompt for the name of the account to be deleted. Given the account name, the processor will confirm that the account does exist. If the account has a password, or if it does not contain the file "/FILE.DEF/", then the account deletion process will be aborted.

Next, the DELETE.ACCOUNT processor executes an INFORM "SELECT" statement, to retrieve from the account's VOC file those Verb definition records which define programs cataloged from the account into the system CATALOG space; these records are identifiable in the VOC file by containing a "V" in the TYPE field (field 1), and having the Verb name (field 2) begin with an asterisk ("*"). (If no such entries exist, the System Administrator will see the message "No records meeting selection criteria found on the VOC file" during this portion of the DELETE.ACCOUNT processing.) Using this SELECT list, the DELETE.ACCOUNT processor executes the "DELETE.CATALOG" command to delete the corresponding programs from the system CATALOG space. This process automatically deletes all programs that have been cataloged from this account during its existence.

The DELETE.ACCOUNT processor next reads the file named "/FILE.DEF/" in the account being deleted, and deletes all the files and dictionaries whose names appear there; any non-PRIME INFORMATION files and sub-UFD's (files and sub-UFD's whose names do not appear in the "/FILE.DEF/" file) are left intact in the account. When this portion of the account deletion process has been completed, the account will be empty of all PRIME INFORMATION files (/FILE.DEF/, VOC, etc.). If no non-PRIME INFORMATION files or sub-UFD's exist in the account, then the account will be completely empty (PRIMOS will see it as a NULL UFD). In this event, the System Administrator may wish to utilize the PRIMOS-level utility "FUTIL", selecting its "TREDEL" option, to delete the empty UFD. (See the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108 and THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109 for information on how to use FUTIL.)

```
    DELETE.ACCOUNT
```

FIGURE 1. Format of the DELETE.ACCOUNT command.

```
:DELETE.ACCOUNT

Acount name =DEL.TEST

No records meeting selection criteria found on the VOC File.

Deleting file VOC
Deleting file D_&UFD&
Deleting file D_VOC
Deleting file &SAVEDLISTS&
Deleting file D_&SAVEDLISTS&
Deleting file VOCLIB
Deleting file D_VOCLIB
The account has been deleted.
:
```

FIGURE 2. Example of the DELETE.ACCOUNT command.

5.28 DELETE.CATALOG Command

The DELETE.CATALOG command makes cataloged programs unavailable for subsequent execution.

The DELETE.CATALOG command marks a cataloged program as unavailable for execution by subsequent calls. The format of this command is:

```
     DELETE.CATALOG {program.name}
```

where "program.name" is the exact name of the program as it appears in the catalog. (See the section titled "MAP Command" in this manual for a command to list the contents of your catalog space.)

The deletion of a program from the catalog does not affect any tasks that are currently executing the program, because the actual cataloged object code is not destroyed by the DELETE.CATALOG command; it is simply marked as ineligible for subsequent invocation. This also means that the space occupied by the deleted code is not made available for use by new cataloged programs until the catalog space reclamation routine is performed via the GARBAGECOLLECT option of the INFORMATION BOOTSTRAP PROCESSOR).

Note that deletion of a subroutine from the catalog must be done with care, and with knowledge of other users on the system. If a cataloged subroutine is in use by a program which has already called that subroutine before the DELETE.CATALOG command is used, the program will continue to execute normally. However, if a program calls a subroutine _after_ it has been deleted from the catalog by the DELETE.CATALOG command (even if the program was already executing before the DELETE.CATALOG command was used to delete the subroutine), the program will produce the error "SUBROUTINE NOT CATALOGED".

```
     DELETE.CATALOG {program.name}
```

FIGURE 1. Format of the DELETE.CATALOG command.

```
:WHO
4 DEMO
:DELETE.CATALOG PGM1

*DEMO*PGM1
Catalog deletion completed.
```

FIGURE 2. Example of the DELETE.CATALOG command (locally cataloged).

```
:WHO
2 MASTER
:DELETE.CATALOG *POSTING

*POSTING
Catalog deletion completed.
```

FIGURE 3. Example of the DELETE.CATALOG command (globally cataloged).

5.29 DELETE.FILE Command

The DELETE.FILE command deletes a file and/or its associated dictionary. This command uses a SELECT list if one is active.

The DELETE.FILE command is used to delete the DATA section, the DICT section, or both sections of a PRIME INFORMATION file. The format of the command is:

```
     DELETE.FILE {DATA} {file.name}

               or

     DELETE.FILE {DICT} {file.name}
```

If the DATA keyword is specified, and field two of the file definition record in the VOC file equals "file.name", then the DATA section of "file.name" will be deleted, and field two in the file definition record will be set to null. If field two of the file definition record does not equal "file.name", the DELETE.FILE processor will ask if it is o.k. to delete the file whose name is in field two. If the reply is no, the file definition record will not be altered. If the reply is yes, the file whose name is in field two of the file definition record will be deleted, and field two will be nulled.

If the DICT keyword is specified and field three of the file definition record in the VOC file equals "D_file.name", the DICT section "D_file.name" will be deleted, and field three in the file definition record will be nulled. If field three of the file definition record does not equal "D_file.name", the DELETE.FILE processor will ask if it is o.k. to delete the file whose name is in field three of the file definition record. If the reply is no, the file definition record will not be altered.

If neither the DATA nor DICT keyword is specified, both sections of "file.name" will be deleted, provided that field two of the file definition record equals "file.name" and field three of the file definition record equals "D_file.name". If fields two and three do not equal "file.name" and "D_file.name", respectively, the DELETE.FILE processor will ask if it is o.k. to delete the file whose name appears in field two; the file definition record in the VOC file will be deleted in either case. If the file being deleted is a REMOTE file, only the VOC file record will be deleted, and the treename which defines the remote file will be displayed.

```
     DELETE.FILE {DATA} {file.name}

               or

     DELETE.FILE {DICT} {file.name}
```

FIGURE 1. Format of the DELETE.FILE Command.

```
:DELETE.FILE DATA PAYROLL
Deleting file PAYROLL, type 2.

:DELETE.FILE DICT CUSTOMERS
Deleting file D_CUSTOMERS, type 3.

:DELETE.FILE INVENTORY
Deleting file INVENTORY, type 7.
Deleting file D_INVENTORY, type 3.
```

FIGURE 2. Examples of the DELETE.FILE Command.

5.3Ø DELETE.LIST Command

Lists of record identifiers that have been constructed by the SELECT or SSELECT commands may be stored and subsequently referenced by name. When these lists are no longer needed, they may be deleted by the DELETE.LIST command.

After the SELECT or SSELECT commands have been used to construct a list of identifiers of records which meet specified selection criteria, the list may be saved by the SAVE.LIST command, for subsequent recall by the GET.LIST command. (See the description of the SELECT and SSELECT commands in THE INFORM REFERENCE GUIDE, IDR39Ø5, and of the SAVE.LIST and GET.LIST commands elsewhere in this manual.) Use of these saved lists offers the advantage of being able to use a given select list more than once without having to re-access the data file in order to reconstruct the list. Eventually, such lists will have out-lived their usefulness, and should be deleted. Deletion of a stored select list may be accomplished through the use of the command:

```
     DELETE.LIST {list.name}
```

This command deletes "list.name" (or "&TEMPport#&" if "list.name" is not specified) from the file &SAVEDLISTS&, where SELECT lists are saved. All saved select lists are stored under the user-specified name "list.name", or the default name "&TEMPport#&", concatenated with a three-digit number "seq", where "seq" is "ØØØ" for saved lists of the short form. If the saved select list is in the long form, groups of up to 5ØØØ record identifiers are stored in separate records. The first group is stored with "seq" equal to "ØØ1", and "seq" is incremented by one for each additional group of record identifiers. This appended three-digit number is never specified with the DELETE.LIST command, since the DELETE.LIST processor automatically provides this number as needed.

```
    DELETE.LIST {list.name}
```

FIGURE 1. Format of the DELETE.LIST Command.

```
:DELETE.LIST ORDERS

:DELETE.LIST BACK.ORDERS

:DELETE.LIST
```

FIGURE 2. Examples of the DELETE.LIST Command.

5.31 DISPLAY Command

During execution of a paragraph it is sometimes desirable to display a line on the terminal to advise the operator of the progress of the entire procedure. This can be accomplished through the use of the DISPLAY command.

The format of the DISPLAY command is as follows:

```
DISPLAY display.text
```

This command causes "display.text" to be printed on the terminal, followed by a carriage-return and a line feed.

```
     DISPLAY display.text
```

FIGURE 1. Format of the DISPLAY Command.

With this Paragraph stored in the VOC file:

```
      id: INV.RUN.1

     001: PA
     002: LIST INV.FILE WITH QOH < 100 LPTR
     003: DISPLAY STARTING INV2203
     004: RUN BP INV2203
     005: DISPLAY PRINTING ERROR.FILE
     006: LIST ERROR.FILE LPTR
```

The following would be the result at the terminal:

```
     :INV.RUN.1
     STARTING INV2203
     PRINTING ERROR.FILE
```

FIGURE 2. Example of the DISPLAY Command.

5.32 ED Command

The ED command invokes the PRIME INFORMATION EDITOR. This command uses a SELECT list if one is active.

The PRIME INFORMATION EDITOR is a line-oriented text editor, used to create, update or delete INFO/BASIC source programs, data and dictionary items, and text files for RUNOFF processing. This EDITOR may be used on records in any PRIME INFORMATION file, regardless of data type or file type. The EDITOR is invoked by the ED command. The format of this command is:

```
    ED                                file.name & record.id's prompted

    ED {DICT} file.name               record.id's prompted

    ED {DICT} file.name rec.id        specified record.id

    ED {DICT} file.name rec.id rec.id ...multiple record.id's

    ED {DICT} file.name *             all records in file
```

The optional keyword "DICT" may be used to specify that the dictionary of the named file is to be edited.

The EDITOR can only access valid PRIME INFORMATION files, i.e., those files created by the PERFORM CREATE.FILE processor.

The PRIME INFORMATION EDITOR is completely documented in THE PRIME INFORMATION EDITOR REFERENCE, IDR3904.

ED	file.name & record.id's prompted
ED {DICT} file.name	record.id's prompted
ED {DICT} file.name rec.id	specified record.id
ED {DICT} file.name rec.id rec.id ...	multiple record.id's
ED {DICT} file.name *	all records in file

FIGURE 1. Format of the ED Command.

5.33 ENTER (INFORM System Processor)

ENTER is a basic data entry processor which belongs to the PRIME INFORMATION system INFORM processor. ENTER is a synonym for ENTRO, which is documented completely in THE INFORM REFERENCE GUIDE, IDR3905. This command uses a SELECT list if one is active.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. ENTER is one of these INFORM processors.

5.34 ENTRO (INFORM System Processor)

ENTRO is a basic data entry processor which belongs to the PRIME INFORMATION system INFORM processor. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905. This command uses a SELECT list if one is active.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. ENTRO is one of these INFORM processors.

5.35 ENTROC (INFORM System Processor)

ENTROC is a basic data entry processor which belongs to the PRIME INFORMATION system INFORM processor. ENTROC is actually a synonym for ENTRO, which is documented completely in THE INFORM REFERENCE GUIDE, IDR3905, except that ENTROC uses cursor addressing. This command uses a SELECT list if one is active.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. ENTROC is one of these INFORM processors.

5.36 FILE.STAT Command

The FILE.STAT command allows the user to obtain useful information regarding the size of a file and the distribution of records within the file.

Information may be obtained about a PRIME INFORMATION file through the use of the FILE.STAT command. This command provides information about the type of the file, its modulo, the quantity of data in the file (in bytes), and its distribution into file groups. The command has the following format:

```
     FILE.STAT {DICT} {file.name} {LPTR}
```

For more detailed information on record distribution within the file, see the description of the GROUP.STAT and GROUP.STAT.DETAIL commands in this manual. For an understanding of the data produced by the FILE.STAT command, see the section titled "PRIME INFORMATION Files" in this manual.

```
FILE.STAT {DICT} {file.name} {LPTR}
```

FIGURE 1. Format of the FILE.STAT Command.

```
:FILE.STAT VOC

File name                           = VOC
File type                           = 5
Number of groups in file (modulo)   = 21
Number of records                   = 276
Total number of bytes               = 22789

Average number of records per group = 13.1428
Average number of bytes per group   = 1085.1904

Average number of bytes per record  = 82.5688
Minimum number of bytes in a record = 15
Maximum number of bytes in a record = 379

Average number of fields per record = 2.9384
Minimum number of fields per record = 2
Maximum number of fields per record = 19
```

FIGURE 2. Example of the FILE.STAT Command.

5.37 FORM.LIST Command

Lists of record identifiers that have been constructed by INFO/BASIC programs may be used by PERFORM as if they had been constructed by the SELECT or SSELECT commands. These lists are retrieved and prepared for processing by the FORM.LIST command.

It may be desirable to write an INFO/BASIC program which produces a list of record identifiers that are to be involved in subsequent processing. Such a program might select records whose keys are to be included in the list, using criteria provided in response to a dialogue with the user at the terminal. Such a list could be stored as a multi-valued record in a PRIME INFORMATION file, with each value being a unique record identifier.

Once stored, a user-built select list may be recalled by the command:

```
     FORM.LIST {file.name} {list.name}
```

The FORM.LIST command retrieves the user-built select list from the file named "file.name" and the record named "list.name", and makes the list available to INFO/BASIC READNEXT statements, to PERFORM commands which use select lists, or to the INFORM processor. A process that makes use of select lists will be unable to distinguish between a select list formed by the SELECT or SSELECT processors, and one recalled by the FORM.LIST command. Indeed, the FORM.LIST command may be thought of as being exactly like the GET.LIST command, since both commands activate a previously saved SELECT list; the difference is that the FORM.LIST command activates a user-built list from a user-specified file, while the GET.LIST command activates a system-built list from the &SAVEDLISTS& file.

See also the sections titled "GET.LIST Command", "SAVE.LIST Command" and "DELETE.LIST Command" in this manual, and the sections on the SELECT and SSELECT commands in THE INFORM REFERENCE GUIDE, IDR3905.

```
    FORM.LIST {file.name} {list.name}
```

FIGURE 1. Format of the FORM.LIST Command.

```
:FORM.LIST KEY.FILE CUSTOMERS.MONDAY

478 records selected

:FORM.LIST MASTER MAIL.LIST

529 records selected
```

FIGURE 2. Examples of the FORM.LIST Command.

5.38 FORMAT Command

The FORMAT command will organize INFO/BASIC source statements into a visible logical block structure for easy reading. This command uses a SELECT list if one is active.

The FORMAT command analyzes an INFO/BASIC source program and re-organizes its physical appearance to reflect its logical structure, utilizing indentation to improve its readability. The format of the command is:

```
     FORMAT {file.name} {program.name} {-LIST}
```

If the single-line form of the command is used (with no parameters), the FORMAT processor will prompt for "file.name" and "program.name". If the "file.name" parameter is specified, but the "program.name" parameter is omitted, the FORMAT processor will not prompt for the "program.name", since it assumes that a SELECT list is present when this form of the command is used.

The use of the -LIST keyword will cause the FORMAT processor to spool the results of the formatting operation directly to the system printer.

For an example of a program before and after using the FORMAT processor, see the section titled "FORMAT Example" in this manual. For an example of the FORMAT command when used with an active SELECT list, see the section titled "Commands Which Use SELECT Lists" in this manual.

```
     FORMAT {file.name} {program.name} {-LIST}
```

The command will prompt for the "file.name" and "program.name" if the one-line format of the command is not used.

FIGURE 1. Format of the FORMAT command.

```
:FORMAT

INFO/BASIC file name     - =MASTER.SOURCE
Spool to printer  (Y/N)  - =N
INFO/BASIC Program name  - =ACCOUNTING.DRIVER
ACCOUNTING.DRIVER
**********************
224 lines formatted.
INFO/BASIC Program name  - = <NEW LINE>
:
```

Note that the same result would be achieved with the single-line form of the command:

```
:FORMAT MASTER.SOURCE ACCOUNTING.DRIVER
```

FIGURE 2. Example of the FORMAT command.

5.38.1 FORMAT Example

These pages present an example of the FORMAT command, by displaying a program whose block structure is difficult to perceive visually, and that same program after the FORMAT processor has produced a highly-visible block structure.

PROGRAM BEFORE FORMATTING

```
*     PRIME NUMBER GENERATOR

LIMIT = 100
DIM PRIME.LIST (LIMIT)
PRIMES.FOUND = 0
FOR NUMBER = 1 TO LIMIT
IF PRIMES.FOUND = 0 THEN STOP.POINT = INT(NUMBER / 2)
ELSE STOP.POINT = PRIME.LIST (PRIMES.FOUND)
FOR DIVISOR = 2 TO STOP.POINT
QUOTIENT = NUMBER / DIVISOR
IF INT(QUOTIENT) = QUOTIENT THEN GOTO NOT.PRIME
NEXT DIVISOR
PRIMES.FOUND += 1
PRINT "*":
PRIME.LIST (PRIMES.FOUND) = NUMBER
IF PRIMES.FOUND => LIMIT THEN GOTO FINISHED

NOT.PRIME:
NEXT NUMBER

FINISHED:
PRINT
FOR NUMBER = 1 TO PRIMES.FOUND
PRINT PRIME.LIST (NUMBER)
NEXT NUMBER

END
```

FIGURE 1. The PRIME.NUMBER program before formatting.

```
:FORMAT EXAMPLES PRIME.NUMBER
 PRIME.NUMBER
 ***
27 lines formatted.
```

FIGURE 2. The FORMAT command used to format this program.

PROGRAM AFTER FORMATTING

```
*      PRIME NUMBER GENERATOR

       LIMIT = 100
       DIM PRIME.LIST (LIMIT)
       PRIMES.FOUND = 0
       FOR NUMBER = 1 TO LIMIT
          IF PRIMES.FOUND = 0 THEN STOP.POINT = INT(NUMBER / 2)
             ELSE STOP.POINT = PRIME.LIST (PRIMES.FOUND)
          FOR DIVISOR = 2 TO STOP.POINT
             QUOTIENT = NUMBER / DIVISOR
             IF INT(QUOTIENT) = QUOTIENT THEN GOTO NOT.PRIME
          NEXT DIVISOR
          PRIMES.FOUND += 1
          PRINT "*":
          PRIME.LIST (PRIMES.FOUND) = NUMBER
          IF PRIMES.FOUND => LIMIT THEN GOTO FINISHED

NOT.PRIME:
       NEXT NUMBER

FINISHED:
       PRINT
       FOR NUMBER = 1 TO PRIMES.FOUND
          PRINT PRIME.LIST (NUMBER)
       NEXT NUMBER

   END
```

FIGURE 3. The PRIME.NUMBER program after formatting.

5.39 GET.LIST Command

Lists of record identifiers that have been constructed by the SELECT and SSELECT processors may be stored and referenced by name. These lists are retrieved and prepared for processing by the GET.LIST command.

After constructing a list of identifiers of records which meet a specified set of selection criteria (with the SELECT or SSELECT commands, described in THE INFORM REFERENCE GUIDE, IDR3905), the list may be stored and subsequently retrieved by name. This technique has the advantage of being able to use a given select list more than once, without having to re-access the data file to reconstruct the list.

The select list may be stored in a file named &SAVEDLISTS&, by using the command SAVE.LIST, described elsewhere in this manual. Once stored, a select list may be recalled by the command:

```
     GET.LIST {account.name} {list.name}
```

The GET.LIST command retrieves the previously-stored select list from the file &SAVEDLISTS&, using the name "list.name", and makes the list available to INFO/BASIC READNEXT statements, to PERFORM commands which use select lists, or to the INFORM processor. A process that makes use of select lists will be unable to distinguish between a select list formed by the SELECT or SSELECT processors, and one recalled by the GET.LIST command.

If the optional "account.name" is specified, the GET.LIST command will retrieve the saved list from that account. Note that this format of the command cannot work if "account.name" requires a password. If no "list.name" is specified, the GET.LIST command will retrieve the list with the name &TEMPport#&, where "port#" is the port number of the user invoking the command.

All saved select lists are stored under the user-specified name "list.name", or under the default name &TEMPport#&, concatenated with a three-digit number "seq", where "seq" is "000" for saved lists of the short form. If the saved select list is in the long form, groups of up to 5000 record identifiers are stored in separate records. The first group is stored with "seq" equal to "001", and "seq" is incremented for each additional group of record identifiers. This appended three-digit number is never specified with the GET.LIST command, since the GET.LIST processor automatically supplies this number as needed.

See also the sections titled "SAVE.LIST Command" and "DELETE.LIST Command" in this manual, and the description of the "&SAVEDLISTS&" file in the Appendix to this manual.

```
     GET.LIST {account.name} {list.name}
```

FIGURE 1. Format of the GET.LIST Command.

```
:GET.LIST CUSTOMERS

3478 records selected.

:GET.LIST MASTER MAIL.LIST

12529 records selected.

:GET.LIST

35 records selected.
```

FIGURE 2. Examples of the GET.LIST Command.

5.40 GROUP.STAT Command

GROUP.STAT is a command that produces a record distribution histogram for a specified file. This command uses a SELECT list if one is active.

The objective of hash-encoded direct access files such as are used for data files in the PRIME INFORMATION system is to evenly distribute the records of the file across a number of groups. With an understanding of the algorithms employed for each of the hash-encoded file types (Types 2 through 9) and knowledge of the actual values of the record identifiers, it is possible to choose a combination of file type and modulo which is likely to produce the best possible distribution of records.

However, when the record identifiers are relatively unpredictable, or as the file grows in size, it is often necessary to view the actual record distribution, to verify that the file type/modulo combination is in fact the correct choice. For this purpose, the GROUP.STAT command has been provided.

The format of the GROUP.STAT command is:

```
     GROUP.STAT {DICT} {file.name} {LPTR}
```

The "DICT" and "file.name" parameters may be omitted as a signal for this verb to use a SELECT list.

The GROUP.STAT verb produces a summary of each group, displaying the number of bytes in the group, the number of records in the group, and a histogram of the number of records in the group. In addition, the GROUP.STAT verb displays the file type description, file type number, and modulo, and produces statistics including total bytes in all groups, total number of records in all groups, average bytes and records per group, and the standard deviation and percent standard deviation of bytes and records per group.

A complement to the GROUP.STAT command is the GROUP.STAT.DETAIL command, described elsewhere in this manual. The GROUP.STAT.DETAIL command provides (as its name implies) more detailed information about the distribution of records in a specified file, including actual record identifiers and corresponding record sizes in each group.

Examples of the GROUP.STAT command are shown in Figures 2 and 3.

```
GROUP.STAT {DICT} {file.name} {LPTR}
```

FIGURE 1. Format of the GROUP.STAT command.

```
:GROUP.STAT CUST

 Type description= Hashed, keys end mainly in numbers
 Bytes  Records          File= CUST    Modulo= 3    Type= 3
   117      3 >>>
   181      2 >>
   229      4 >>>>
======   ====
   527      9  Totals
   176      3  Averages per group
    45      0  Standard deviation from average
  25.6    0.0  Percent std dev from average
```

FIGURE 2. Example of GROUP.STAT command.

```
:GROUP.STAT PATIENTS

Type description= Hashed, keys end in full range of ASCII characters
 Bytes  Records          File= PATIENTS    Modulo= 7    Type= 5
  1347      5 >>>>>
   962      3 >>>
  1263      5 >>>>>
  1325      5 >>>>>
   808      3 >>>
   693      3 >>>
  2059      5 >>>>>
======   ====
  8457     29  Totals
  1208      4  Averages per group
   421      0  Standard deviation from average
  34.9    0.0  Percent std dev from average
```

FIGURE 3. Example of GROUP.STAT command.

5.41 GROUP.STAT.DETAIL Command

GROUP.STAT.DETAIL is a command that produces a detailed record distribution summary for a specified file. This command uses a SELECT list if one is active.

The objective of hash-encoded direct access files such as are used for data files in the PRIME INFORMATION system is to evenly distribute the records of the file across a number of groups. With an understanding of the algorithms employed for each of the hash-encoded file types (Types 2 through 9) and knowledge of the actual values of the record identifiers, it is possible to choose a combination of file type and modulo which is likely to produce the best possible distribution of records.

However, when the record identifiers are relatively unpredictable, or as the file grows in size, it is often necessary to view the actual record distribution, to verify that the file type/modulo combination is in fact the correct choice. For this purpose, the GROUP.STAT.DETAIL command has been provided.

The format of the GROUP.STAT.DETAIL command is:

```
     GROUP.STAT.DETAIL {DICT} {file.name} {LPTR}
```

The "DICT" and "file.name" parameters may be omitted as a signal for this verb to use a SELECT list.

The GROUP.STAT.DETAIL command produces a detailed summary of each group, displaying the record id and size in bytes of each record in the group, and producing a total line for each group, which includes the size of the group in bytes, and the number of records in the group. In addition, the GROUP.STAT.DETAIL verb displays the file type description, file type number, and modulo, and produces statistics including total bytes in all groups, total number of records in all groups, average bytes and records per group, and the standard deviation and percent standard deviation of bytes and records per group.

A functionally similar record distribution display, but with less detail and with each group represented by a histogram, is available from the GROUP.STAT command, described elsewhere in this manual.

An example of the GROUP.STAT.DETAIL command is shown in Figure 2.

```
     GROUP.STAT.DETAIL {DICT} {file.name} {LPTR}
```

FIGURE 1. Format of the GROUP.STAT.DETAIL command.

```
:GROUP.STAT.DETAIL CUST
 Type description= Hashed, keys end mainly in numbers
 Bytes  Record.id          File= CUST    Modulo= 3    Type= 3
    24   1Ø1.A
    48   1Ø3.B
    39   1Ø4.A
Group 1 Total size: 117 Bytes    3 Records

   114   1Ø1.B
    61   1Ø5.A
Group 2 Total size: 181 Bytes    2 Records

    46   1Ø3.D
    61   1Ø2.B
    41   1Ø6.A
    75   1Ø3.A
Group 3 Total size: 229 Bytes    4 Records

Bytes  Records
   527      9  Totals
   176      3  Averages per group
    45      Ø  Standard deviation from average
  25.6    Ø.Ø  Percent std dev from average
```

FIGURE 2. Example of GROUP.STAT.DETAIL command.

5.42 HASH.HELP Command

HASH.HELP is a command that analyzes an existing PRIME INFORMATION file, looking closely at record id characteristics and record id/record sizes, to recommend a file type and modulo for that file. This command uses a SELECT list if one is active.

The HASH.HELP verb is provided as an aid to determining the best file type and modulo for an existing PRIME INFORMATION file. Since hash-encoded direct access files (used for data files on PRIME INFORMATION systems) may affect overall system performance according to how well the records in them are distributed, it is important to choose a combination of file type and modulo which optimizes this distribution. The HASH.HELP verb is intended to assist a knowledgable System Administrator in determining what is the "best" file type and modulo which will assure the most even distribution of records in the file.

The format of the HASH.HELP command is:

```
     HASH.HELP {DICT} {file.name} {NO.PAGE} {LPTR}
```

The "DICT" and "file.name" parameters may be omitted as a signal for this verb to use a SELECT list, or (if no SELECT list is active) to prompt for "file.name". Since HASH.HELP is an aid in hash-encoding files more efficiently, Type 1 files (which are not hash-encoded) may not be specified by the "file.name" parameter. If a SELECT list is used, but the terminal being used is a printing device, you may wish to use the NO.PAGE parameter. The report produced by this verb may be routed to the system printer by using the LPTR parameter.

The HASH.HELP verb analyzes a file by reading every key and every record in the file. The record id's are examined carefully to determine their predominant data type, and to determine whether they tend to be more unique from left-to-right, or from right-to-left. This information, combined with average record id and record data sizes, is used as the basis for a recommendation as to "best" file type and "minimum" modulo. Since this process depends on averages, the recommendations produced are just that: recommendations. In some cases, the file type recommended may not actually be the optimum one, and it is likely that the modulo recommended (since it is a <u>minimum</u> modulo recommendation) will usually be smaller than is ideal.

A complement to the HASH.HELP command is the HASH.HELP.DETAIL command, described elsewhere in this manual. The HASH.HELP.DETAIL command provides (as its name implies) more detailed information about the distribution of records in a specified file, including minimum, maximum, and average record id and record data sizes.

```
HASH.HELP {DICT} {file.name} {NO.PAGE} {LPTR}
```

FIGURE 1. Format of the HASH.HELP command.

```
:HASH.HELP STATES

File  STATES  Type 6  Modulo 2      16:32:04  10-10-80  Page 1

Of the 51 total keys in this file:

    0  keys were wholly numeric (digits 0 thru 9)
       (Use File Type 2 or 6 for wholly numeric keys)

    0  keys were numeric with separators (as reproduced below):
       0123456789#$%&*+-./:;_
       (Use File Type 3 or 7 for numeric keys with separators)

   51  keys were from the 64-character ASCII set reproduced below:
       !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
      (Use File Type 4 or 8 for 64-character ASCII keys)

    0  keys were from the 256-character ASCII set
       (Use File Type 5 or 9 for 256-character ASCII keys)

The keys in this file are more unique in their right-most eight bytes.
The smallest modulo you should consider for this file is 4.
The best type to choose for this file is probably type 4.
```

FIGURE 2. Example of the HASH.HELP command.

5.43 HASH.HELP.DETAIL Command

HASH.HELP.DETAIL is a command that analyzes an existing PRIME INFORMATION file, looking closely at record id characteristics and record id/record sizes, to recommend a file type and modulo for that file. This command uses a SELECT list if one is active.

The HASH.HELP.DETAIL verb is provided as an aid to determining the best file type and modulo for an existing PRIME INFORMATION file. Since hash-encoded direct access files (used for data files on PRIME INFORMATION systems) may affect overall system performance according to how well the records in them are distributed, it is important to choose a combination of file type and modulo which optimizes this distribution. The HASH.HELP.DETAIL verb is intended to assist a knowledgable System Administrator in determining what is the "best" file type and modulo which will assure the most even distribution of records in the file.

The format of the HASH.HELP.DETAIL command is:

```
     HASH.HELP.DETAIL {DICT} {file.name} {NO.PAGE} {LPTR}
```

The "DICT" and "file.name" parameters may be omitted as a signal for this verb to use a SELECT list, or (if no SELECT list is active) to prompt for "file.name". Since HASH.HELP.DETAIL is an aid in hash-encoding files more efficiently, Type 1 files (which are not hash-encoded) may not be specified by the "file.name" parameter. If a SELECT list is used, but the terminal being used is a printing device, you may wish to use the NO.PAGE parameter. The report produced by this verb may be routed to the system printer by using the LPTR parameter.

The HASH.HELP.DETAIL verb analyzes a file by reading every key and every record in the file. The record id's are examined carefully to determine their predominant data type, and to determine whether they tend to be more unique from left-to-right, or from right-to-left. This information, combined with average record id and record data sizes, is used as the basis for a recommendation as to "best" file type and "minimum" modulo. Since this process depends on averages, the recommendations produced are just that: recommendations. In some cases, the file type recommended may not actually be the optimum one, and it is likely that the modulo recommended (since it is a <u>minimum</u> modulo recommendation) will usually be smaller than is ideal.

A complement to the HASH.HELP.DETAIL command is the HASH.HELP command, described elsewhere in this manual. The HASH.HELP command provides more concise information about the distribution of records in a specified file, omitting minimum, maximum, and average record id and record data sizes.

```
HASH.HELP.DETAIL {DICT} {file.name} {NO.PAGE} {LPTR}
```

FIGURE 1. Format of the HASH.HELP.DETAIL command.

```
:HASH.HELP.DETAIL STATES

File  STATES   Type 6   Modulo 1       16:41:19  10-10-80   Page 1

       51  total keys in file
      102  total bytes in all keys
        2  smallest key
        2  largest key
        2  average bytes per key

       51  total records in file
     5514  total bytes in all records
       86  smallest record
      122  largest record
   108.00  average bytes per record

       17  average records per group  (optimized)
     1838  average bytes per group  (optimized)

Press <NEW LINE> to continue =<NEW LINE>

File  STATES   Type 4   Modulo 3       16:41:19  10-10-80   Page 2

Of the 51 total keys in this file:

    0  keys were wholly numeric (digits 0 thru 9)
       (Use File Type 2 or 6 for wholly numeric keys)

    0  keys were numeric with separators (as reproduced below):
       0123456789#$%&*+-./:;_
       (Use File Type 3 or 7 for numeric keys with separators)

   51  keys were from the 64-character ASCII set reproduced below:
       !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
       (Use File Type 4 or 8 for 64-character ASCII keys)

    0  keys were from the 256-character ASCII set
       (Use File Type 5 or 9 for 256-character ASCII keys)

The keys in this file are more unique in their right-most eight bytes.
The smallest modulo you should consider for this file is 4.
The best type to choose for this file is probably type 4.
```

FIGURE 2. Example of the HASH.HELP.DETAIL command.

5.44 HASH.TEST Command

HASH.TEST is a command that produces a record distribution histogram for an existing PRIME INFORMATION file, with the user supplying a new file type and/or modulo. This command uses a SELECT list if one is active.

The HASH.TEST command is used to give a "preview" of the distribution of records in an existing PRIME INFORMATION file, given a new file type and/or modulo. The format of the HASH.TEST command is:

```
     HASH.TEST {DICT} {file.name} {type} {modulo} {LPTR}
```

The "DICT" and "file.name" parameters may be omitted as a signal for this verb to use a SELECT list. The "type" and "modulo" parameters are the same as for the CREATE.FILE command, except that Type 1 is not acceptable for the HASH.TEST command. An asterisk (*) for either of these parameters is a signal to use the existing type or modulo, respectively. Additionally, both the "type" and the "modulo" parameters may be expressed as if they were control variables for a loop. In this format (taking the "type" parameter as an example), file "type" could be expressed as:

```
     begin.type,end.type,increment
```

where "begin.type" is the first file type to be tried (and must be an integer between 2 and 9); end.type is the last file type to be tried (and must also be an integer between 2 and 9); and "increment" must be an integer. As an example, the command:

```
     HASH.TEST INVENTORY.MASTER 2,6,2 *
```

would produce a trial histogram for the "INVENTORY.MASTER" file using its current "modulo" (signified by the asterisk), and using first file type 2, then file type 4, then file type 6.

The HASH.TEST verb produces a summary of each group, displaying a histogram of the number of records in the group. In addition, the HASH.TEST verb displays the file type description, file type number, and modulo, and produces statistics including total number of records in all groups, average records per group, and the standard deviation and percent standard deviation of records per group.

A complement to the HASH.TEST command is the HASH.TEST.DETAIL command, described elsewhere in this manual. The HASH.TEST.DETAIL command provides (as its name implies) more detailed information about the distribution of records in a specified file, including corresponding record sizes in each group.

```
    HASH.TEST {DICT} {file.name} {type} {modulo} {LPTR}
```

FIGURE 1. Format of the HASH.TEST command.

```
:HASH.TEST STATES * *

 Type description= Hashed, keys end in letters
        records              file= STATES    modulo= 1    type= 4
          51 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
        ====
          51  totals
          51  averages per group
           0  standard deviation from average
         0.0  percent std dev from average
:HASH.TEST STATES 3 3

 Type description= Hashed, keys end mainly in numbers
       records              file= STATES    modulo= 3    type= 3
          21 >>>>>>>>>>>>>>>>>>>>>
          21 >>>>>>>>>>>>>>>>>>>>>
           9 >>>>>>>>>
        ====
          51  totals
          17  averages per group
           5  standard deviation from average
        29.4  percent std dev from average
:HASH.TEST STATES 7 5

 Type description= Hashed, keys begin mainly in numbers
       records              file= STATES    modulo= 5    type= 7
           8 >>>>>>>>
          14 >>>>>>>>>>>>>>
           2 >>
           9 >>>>>>>>>
          18 >>>>>>>>>>>>>>>>>>
        ====
          51  totals
          10  averages per group
           5  standard deviation from average
        50.0  percent std dev from average
```

FIGURE 2. Examples of the HASH.TEST command.

5.45 HASH.TEST.DETAIL Command

HASH.TEST.DETAIL is a command that produces a record distribution histogram for an existing PRIME INFORMATION file, with the user supplying a new file type and/or modulus. This command uses a SELECT list if one is active.

The HASH.TEST.DETAIL command is used to give a "preview" of the distribution of records in an existing PRIME INFORMATION file, given a new file type and/or modulo. The format of the HASH.TEST.DETAIL command is:

```
HASH.TEST.DETAIL {DICT} {file.name} {type} {modulo} {LPTR}
```

The "DICT" and "file.name" parameters may be omitted as a signal for this verb to use a SELECT list. The "type" and "modulo" parameters are the same as for the CREATE.FILE command, except that Type 1 is not acceptable for the HASH.TEST.DETAIL command. An asterisk (*) for either of these parameters is a signal to use the existing type or modulo, respectively. Additionally, both the "type" and the "modulo" parameters may be expressed as if they were control variables for a loop. In this format (taking the "type" parameter as an example), file "type" could be expressed as:

```
begin.type,end.type,increment
```

where "begin.type" is the first file type to be tried (and must be an integer between 2 and 9); end.type is the last file type to be tried (and must also be an integer between 2 and 9); and "increment" must be an integer. As an example, the command:

```
HASH.TEST.DETAIL INVENTORY.MASTER 3,8,2 *
```

would produce a trial histogram for the "INVENTORY.MASTER" file using its current "modulo" (signified by the asterisk), and using first file type 3, then file type 5, then file type 7.

The HASH.TEST.DETAIL verb produces a summary of each group, displaying a histogram of the number of records in the group. In addition, the HASH.TEST.DETAIL verb displays the file type description, file type number, and modulo, and produces statistics including total number of records in all groups, average records per group, and the standard deviation and percent standard deviation of records per group.

A complement to the HASH.TEST.DETAIL command is the HASH.TEST command, described elsewhere in this manual. The HASH.TEST.DETAIL command differs from the HASH.TEST command in that it provides the size (in bytes) of each group, in addition to the record distribution histogram for each group.

```
HASH.TEST.DETAIL {DICT} {file.name} {type} {modulo} {LPTR}
```

FIGURE 1. Format of the HASH.TEST.DETAIL command.

```
:HASH.TEST.DETAIL STATES * *

 Type description= Hashed, keys end in letters
 bytes  records             file= STATES    modulo= 1    type= 4
  6458     51 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
 ======  ====
  6458     51  totals
  6458     51  averages per group
            0  standard deviation from average
          0.0  percent std dev from average
:HASH.TEST.DETAIL STATES 3 3

 Type description= Hashed, keys end mainly in numbers
 bytes  records             file= STATES    modulo= 3    type= 3
  2695     21 >>>>>>>>>>>>>>>>>>>>>
  2663     21 >>>>>>>>>>>>>>>>>>>>>
  1112      9 >>>>>>>>>
 ======  ====
  6470     51  totals
  2157     17  averages per group
   738      5  standard deviation from average
  34.2   29.4  percent std dev from average
:HASH.TEST.DETAIL STATES 7 5

 Type description= Hashed, keys begin mainly in numbers
 bytes  records             file= STATES    modulo= 5    type= 7
  1021      8 >>>>>>>>
  1818     14 >>>>>>>>>>>>>>
   274      2 >>
  1121      9 >>>>>>>>>
  2248     18 >>>>>>>>>>>>>>>>>>
 ======  ====
  6482     51  totals
  1296     10  averages per group
   682      5  standard deviation from average
  52.6   50.0  percent std dev from average
```

FIGURE 2. Examples of the HASH.TEST.DETAIL command.

5.46 HELP Command

The HELP command invokes the PRIME INFORMATION on-line user assistance processor.

The HELP command is used to request assistance in using the PRIME INFORMATION system. With a system as comprehensive as PRIME INFORMATION, it is easy to forget how to use some infrequently-used commands, or indeed, to forget what they are for. The HELP command is provided to give you enough information about any element which is contained in the VOC file, to be able to use that element (ideally), or, at the least, to know where to look for additional information.

The format of the HELP command is:

```
     HELP {VOC.item.name}
```

If you know the name of the item that you need some HELP in using, specify its name on the command line with the HELP command. For example, if you are having difficulty remembering the order of the parameters required by the COPY command, simply type:

```
     :HELP COPY
```

You will see a description of the COPY command which is a summary of the information available from this manual. However, if you simply need HELP with the entire PRIME INFORMATION system, type HELP, and the HELP processor will ask you to type either a single processor name (the name of an item in the VOC file), or simply <NEW LINE> to view every HELP item in the SYS.HELP and USER.HELP files.

The PRIME INFORMATION system is supplied with HELP items for PRIME-supplied routines in a file named SYS.HELP. Your system administrator may augment this system HELP information by providing HELP information for your own installation's programs, files, commands, sentences, etc., by placing HELP descriptions into a file named USER.HELP, using the same format as the SYS.HELP file.

You may also invoke the HELP processor by typing "?VOC.item.name" at PERFORM's ":" prompt. If you do so, you will be shown the comment portion of Field One of the VOC item you named, and will be given the opportunity to see the SYS.HELP description (and the USER.HELP description, if available) if you wish to do so.

```
     HELP {VOC.item.name}
```

FIGURE 1. Format of the HELP Command.

```
:HELP WHO

DOCUMENTATION AVAILABLE ON SYSTEM HELP FILE FOR   *** WHO ***

Type "WHO"

This command displays the terminal number and the account name
for your task.

Example:

      :WHO
      2 MASTER

In this example, the user is using terminal port number 2, and is
in the INFORMATION account named MASTER.
```

FIGURE 2. Example of the HELP Command.

5.47 HUSH Command

The HUSH command is used to suppress all messages which might normally be sent to a terminal during processing. This command may also be used to re-enable message transmission after it has been suppressed by earlier use of this command.

Sometimes it is desirable to be able to run a task with all terminal output suppressed. For example, a paragraph which is known to produce considerable amounts of display data (in addition to the primary results that the paragraph should produce, which might be a report on the system printer) may be run on a slow device. The slow device could be a CRT terminal operating at slow speed over a telephone line, or it could be a hard-copy terminal which is inherently slow. If the data to be displayed on the terminal is not necessary (it is not the principal product of executing the paragraph), it may be completely suppressed with the HUSH command. The format of the command is:

```
     HUSH {OFF}

         or

     HUSH {ON}
```

Normal system operation is as if the command HUSH ON had been used; i.e., any message destined for the terminal display device is transmitted for display on that device. HUSH OFF will suppress all messages destined for the terminal display device.

This command should be used only when the terminal messages are known to be unnecessary. For example, any request for input data will not appear on the terminal after HUSH OFF has been executed; neither will any error messages (which may well have been unexpected) appear.

HUSH OFF suppresses all messages destined for the terminal, while HUSH ON re-enables terminal display. If neither keyword (ON or OFF) is used with the HUSH command, the command becomes a toggle: if terminal display was in effect, HUSH will disable it; if terminal display was suppressed, HUSH will re-enable it.

This command should not be confused with the command which sets a terminal's display characteristics as either half-duplex or full-duplex. For this requirement, see the description of the TERM command in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, and the description of the AMLC command in THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109.

```
HUSH {OFF}

    or

HUSH {ON}
```

FIGURE 1. Format of the HUSH command.

5.48 IAM Command

The IAM command specifies to PERFORM the name of the PRIME INFORMATION account which should be used by commands which require an account name.

Many PERFORM commands require an account name for proper operation; some examples are CATALOG, which uses the account name along with the program name when a program is cataloged as a local resource, and RELEASE, which releases file and/or record locks for a particular account.

When the user first enters the PRIME INFORMATION environment (either by logging in or by typing INFO while in the PRIMOS environment), PERFORM sets the account name at that instant as the name of the current UFD. When a user switches from one account to another within the PRIME INFORMATION environment (by using the LOGTO command, described elsewhere in this manual), PERFORM automatically uses the name of the new account as the account name. If this is not desired, the account name to be maintained by PERFORM may be specified by using the IAM command.

The format of the IAM command is:

```
      IAM account.name
```

where "account.name" is the account name that PERFORM should use when a name is required (and is normally the name of the account the user is presently attached to).

The WHO command (described elsewhere in this manual) may be used at any time to verify that the current account name is as intended.

```
    IAM account.name
```

FIGURE 1. Format of the IAM command.

```
:WHO
2 DEMO
:IAM MASTER
:WHO
2 MASTER
```

FIGURE 2. Example of the IAM command.

5.49 INFO Command (PRIMOS-level Command)

The INFO command is a PRIMOS-level command which allows the user to leave the PRIMOS environment and enter the PRIME INFORMATION environment.

A user can go from the PRIMOS environment to the PRIME INFORMATION environment by invoking the INFO command. The INFO command is a PRIMOS-level command which has no meaning within the PRIME INFORMATION environment. It may only be invoked at the PRIMOS "OK, " prompt. The format of the INFO command is:

```
INFO {-PHANTOM}
```

The -PHANTOM keyword is a signal to PERFORM that the task invoking INFORMATION is a phantom task. With this keyword, PERFORM will be able to accept all "terminal" input (commands and data) from a COMINPUT file. (See the COMINPUT command description in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, for a description of this command file.) Note that you should include "OFF" as the last command in your COMINPUT file. The PHANTOM command (described elsewhere in this manual) uses this optional keyword in the command file it builds to accomplish the operation specified with the PHANTOM command.

If the INFO command is executed within an account which is not currently a PRIME INFORMATION account, PERFORM displays the message:

```
This UFD is not a valid INFORMATION account
Do you wish to make it one? =
```

If the reply to this question is YES, PERFORM will create in the account all of the files necessary to make it a PRIME INFORMATION account. If the reply to this question is NO, PERFORM will treat it as a QUIT command (see the section titled "QUIT Command" in this manual), allowing the user to re-enter the PRIMOS environment in the account.

If the account has an owner password, and it is omitted or incorrectly specified when the account is ATTACHed to, PERFORM will be unable to identify the account as a PRIME INFORMATION account, whether it is or is not. In this circumstance, the same message as above will be produced. However, PERFORM will (with an incorrect or omitted password) be attached to the new account with non-owner rights, and those rights will determine whether PERFORM will be allowed to make the account a PRIME INFORMATION account, or not. In general, after a password error on the INFO command, it may be necessary to re-specify the INFO command with the correct password, or to leave the PRIME INFORMATION environment, attach to the account with the correct password, and re-enter the PRIME INFORMATION environment (with the INFO command).

```
    INFO
```

FIGURE 1. Format of the INFO command.

```
LOGIN OLDACC
OK, INFO
OLDACC
Release number 4.0
:
```

FIGURE 2. Example of the INFO command (existing account).

```
LOGIN OLDACC
OK, A MFD XXXXXX 1
OK, CR NEWONE
OK, A NEWONE
OK, INFO
NEWONE
This UFD is not a valid information account.
Do you wish to make it one? YES
Creating file &SAVEDLISTS& / Type 1 / Modulo 1.
Creating file D_&SAVEDLISTS& (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_&SAVEDLISTS&.
Creating file VOCLIB / Type 2 / Modulo 7.
Creating file D_VOCLIB (DICT) / Type 3 / Modulo 1.
Added "@ID", the default record for INFORM, to DICT D_VOCLIB.
Release number 4.0
:
```

FIGURE 3. Example of the INFO command (new account).

5.50 ISTAT Command

ISTAT is a command that produces a record distribution histogram for a specified file. ISTAT is a synonym for the command GROUP.STAT. This command uses a SELECT list if one is active.

The ISTAT command is used to monitor the distribution of records within PRIME INFORMATION files. ISTAT is an synonym for the GROUP.STAT command. For a complete description of the ISTAT command, see the description of GROUP.STAT in the section of this manual titled "GROUP.STAT Command".

5.51 LIST (INFORM System Processor)

LIST is the high-level, natural language QUERY and REPORT GENERATOR portion of the PRIME INFORMATION system. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905. This command uses a SELECT list if one is active.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. LIST is one of these INFORM processors.

5.52 LIST.LOCKS Command

The LIST.LOCKS command displays the status of the 64 task synchronization locks.

PRIME INFORMATION CPU's have 64 hardware semaphores which are referred to as execution locks. They may be used to synchronize processes running on different terminals.

The current status of the 64 task synchronization locks can be displayed using the LIST.LOCKS command. The format of this command is:

```
     LIST.LOCKS
```

This command displays the lock number and an integer showing the status of that lock. A status of -1 means that the lock is clear (not set) whereas a status of zero or greater means the lock is set.

See also the description of the CLEAR.LOCKS and LOCK commands elsewhere in this manual. The System Administrator may also wish to review the MASTER command, documented in this manual.

```
LIST.LOCKS
```

FIGURE 1. Format of the LIST.LOCKS command.

```
:LIST.LOCKS
 0: -1 !  1: -1 !  2: -1 !  3: -1 !  4: -1 !  5: -1 !  6: -1 !  7: -1 !
 8: -1 !  9: -1 ! 10: -1 ! 11: -1 ! 12: -1 ! 13: -1 ! 14: -1 ! 15: -1 !
16:  0 ! 17: -1 ! 18: -1 ! 19: -1 ! 20: -1 ! 21: -1 ! 22: -1 ! 23: -1 !
24: -1 ! 25: -1 ! 26: -1 ! 27: -1 ! 28: -1 ! 29: -1 ! 30: -1 ! 31: -1 !
32: -1 ! 33: -1 ! 34: -1 ! 35: -1 ! 36: -1 ! 37: -1 ! 38: -1 ! 39: -1 !
40: -1 ! 41: -1 ! 42: -1 ! 43: -1 ! 44: -1 ! 45: -1 ! 46: -1 ! 47: -1 !
48: -1 ! 49: -1 ! 50: -1 ! 51: -1 ! 52: -1 ! 53: -1 ! 54: -1 ! 55: -1 !
56: -1 ! 57: -1 ! 58: -1 ! 59: -1 ! 60: -1 ! 61: -1 ! 62: -1 ! 63: -1 !
```

FIGURE 2. Example of the LIST.LOCKS command.
(Note that lock number 16 is set.)

5.53 LIST.READU Command

The LIST.READU command is used to request a list of records which have been locked by the INFO/BASIC READU, READVU, or MATREADU commands.

The LIST.READU command is used to request a list of records which have been locked by the INFO/BASIC READU, READVU, or MATREADU commands. The format of the LIST.READU command is:

```
     LIST.READU user.number

         or

     LIST.READU ALL
```

where "user.number" is the task number of the user whose locked records will be displayed on the terminal; the parameter "user.number" must be between 0 and 63. If the "ALL" option is used in place of a task number, then all record locks on the system set by INFO/BASIC READU, READVU, or MATREADU statements will be displayed on the terminal.

For each record locked, the LIST.READU command will display RECORD ID, FILE ID, and LINK fields, as well as the task number of the user for whom the record is locked. The RECORD ID field is an abbreviation of the actual identifier of the record that is locked; this abbreviation is used internally by INFO/DMS in its record lock management routines. In many cases, this RECORD ID field may be identical to the complete record identifier, but it will not be identical in the case of lengthy record identifiers. The FILE ID field contains a number that is used internally by INFO/DMS to identify the file where the locked record resides. (Note that the FILE ID field is not the file name, which is not retained by INFO/DMS after the file is opened.) The LINK field is reserved for future use, and the value that appears there is not meaningful at this time.

```
     LIST.READU user.number

         or

     LIST.READU ALL
```

where:

"user.number"	is the user number whose record locks are to be displayed.
"ALL"	means to display the record locks of all users.

FIGURE 1. Format of the LIST.READU command.

```
:LIST.READU 5
RECORD ID        FILE ID          LINK   FOR USER     5
00010           000400035171        0
TIME            141320120240       13


:LIST.READU ALL
RECORD ID        FILE ID          LINK   FOR USER     5
00010           000400035171        0
TIME            141320120240       13

RECORD ID        FILE ID          LINK   FOR USER     3
074-99-8123     234400235171        0
ON.HAND         141320112345        6
```

FIGURE 2. Examples of the LIST.READU command.

5.54 LO Command

The LO command is an abbreviation for the LOGOUT command.

The LO command is an abbreviation for the LOGOUT command, which is the command which is used to give up access to the system. The LOGOUT command is described in the section titled "LOGOUT Command", elsewhere in this manual.

5.55 LOCK Command

The LOCK command sets the status of any of the 64 task synchronization locks, to the locked state.

PRIME INFORMATION systems have 64 user-accessible semaphores which are referred to as execution locks. They may be used to synchronize processes running on different terminals on the same PRIME INFORMATION system. The current status of any of the 64 task synchronization locks can be set to the locked state by using the LOCK command. The format of this command is:

```
     LOCK semaphore.number {PROMPT}

               or

     LOCK semaphore.number {NO.WAIT}
```

where "semaphore.number" is an integer from 0 through 63, corresponding to the number of the semaphore to be set to the locked state. If the optional "PROMPT" parameter is present, the LOCK processor will give you an opportunity to execute other commands while waiting (if necessary) for the specified semaphore to be unlocked by another user.

If the "PROMPT" parameter is omitted, and the specified "semaphore.number" is already locked by another user, the LOCK command normally will wait until this lock is released before returning to PERFORM's colon prompt. However, you may specify that this wait is not to take place, even if the semaphore is already locked by another user, by using the optional "NO.WAIT" parameter. (Note that neither the "PROMPT" nor the "NO.WAIT" parameters should be used within a PERFORM paragraph stored in the VOC file, since either parameter could cause a prompt for terminal input.)

The LOCK command is a PERFORM-level version of the INFO/BASIC LOCK statement, and the result of both the command and the statement is the same. Usually, task synchronization locks are set within INFO/BASIC programs, but the LOCK command also allows them to be set from the terminal.

See also the description of the LIST.LOCKS and CLEAR.LOCKS commands elsewhere in this manual, and the description of the LOCK statement in THE INFO/BASIC REFERENCE GUIDE, PDR3903. The System Administrator may also wish to review the MASTER command, described in this manual.

WARNING: The task synchronization locks are local to any particular PRIME INFORMATION system. Specifically, they are not effective over a network, including PRIMENET. If you have two or more systems operating together in a networked environment, you cannot use these locks to synchronize events in both systems.

```
     LOCK semaphore.number {PROMPT}

               or

     LOCK semaphore.number {NO.WAIT}
```

FIGURE 1. Format of the LOCK command.

```
:LOCK 3


:LOCK 48 NO.WAIT


:LOCK 13 PROMPT
Enter commands to perform while waiting on lock, or END to wait.
Enter QUIT to give up. I will tell you when the LOCK succeeds.
>WHO
4 KOVSKY
>QUIT
***  LOCK aborted at Operator request.
```

FIGURE 2. Examples of the LOCK command.

5.56 LOGIN Command (PRIMOS-level Command)

To begin an operating session on the PRIME INFORMATION System a user must LOGIN to an account.

LOGIN is the command the user must type at the terminal to begin an operating session with the PRIME INFORMATION System.

```
LOGIN account.name {password} {ldisk} {-ON nodename}
```

"account.name" must be a valid account (UFD) name on any of the disks available to the system. "password" is an optional argument which specifies the password to the account. "ldisk" is an optional argument which specifies the logical disk to search for "account.name". If "ldisk" is not specified all logical disks will be searched starting with logical disk number Ø until "account.name" is found or there are no more logical disks to search.

If the user's system is connected via a network to other PRIME INFORMATION Systems, it is possible to log into an account that is not local to the system to which the user terminal is connected. This is accomplished by use of the -ON option. See LOGIN in the PRIMOS COMMANDS REFERENCE GUIDE, FDR31Ø8, for additional details.

When LOGIN is successful, the user is attached to the account specified by "account.name". A login message is printed at the user terminal and at the supervisor terminal. See example in Figure 2.

A user may LOGTO other accounts during a session but the LOGIN "account.name" is remembered and printed on LOGOUT regardless of which account a user is currently attached to. The LOGIN "account.name" is also printed by the STATUS command.

Most PRIME INFORMATION Systems will be set up to take the user directly to the PERFORM colon prompt (:). If your account is not set up this way, you must enter "INFO", after logging in to go from the PRIMOS prompt of "OK," to the PERFORM prompt of ":".

Any time a PRIME INFORMATION account is initially entered (as in a LOGIN, a LOGTO, or after typing INFO at the PRIMOS "OK," prompt), a VOC file record with an id of "LOGIN" is executed (if there is one). See the section titled "LOGIN Procedure in VOC File" in this manual for additional information about this feature.

```
     LOGIN account.name {password} {ldisk} {-ON nodename}
```

where

> "account.name" is a valid account (UFD) on one of the logical disks.
>
> "password" is the password which must be specified if the account has one.
>
> "ldisk" is the logical disk to searched for "account.name". If omitted, all logical disks will be searched (starting with logical disk Ø).
>
> "-ON nodename" is the remote system a user wishes to log into if your system is part of a PRIMENET network of PRIME INFORMATION Systems.

FIGURE 1. Format of the LOGIN command.

<u>LOGIN JONESY SECRET</u>

This command sequence logs in the user and attaches the user to account "JONESY" (if the password "SECRET" is correct). The system response to this login at the user terminal and at the supervisor terminal is:

```
     JONESY (4) LOGGED IN AT 12'39  Ø3238Ø
```

The number in parentheses is the user number of the user terminal (e.g., 4). The next number is the time of day (e.g., 12:39 PM) and the last number is the date in the form "mmddyy" (e.g., Ø3-23-8Ø).

FIGURE 2. Example of the LOGIN command.

5.57 LOGOUT Command

The last command the user issues to terminate access to the PRIME INFORMATION System is LOGOUT.

LOGOUT is the last command the user issues when giving up access to the PRIME INFORMATION system. The format of this command is:

```
     LOGOUT {-user.number}
```

When the LOGOUT command is invoked, all user files are closed, all devices ASSIGNed to the user terminal are released and a logout message is printed at the user terminal and at the supervisor terminal. See example in FIGURE 2. Additionally, the PERFORM sentence stack is erased (if the item named STACKWRITE in the user's VOC file has "OFF" in field two), or is preserved until the next session (if the item named STACKWRITE in the user's VOC file does not have "OFF" in field two). (See the section titled "PERFORM Sentence Stack Retention (STACKWRITE)" in this manual for more information on the preservation of the sentence stack.)

If a user has started a phantom process and wishes to abort it, the user may log it out using the "-user.number" option, provided the user "-user.number" has the same login name as the user giving the command. "-user.number" is the user number of the user to be logged out. (At the system terminal, the user specified by "-user.number" will be logged out regardless of login name.)

CAUTION: if you LOGOUT a task other than the one running at your terminal, the data base may be adversely affected. "LOGOUT -user.number" is an immediate command, and overrides all protections of INFO/DMS against damaging the data base by only partially completing a disk record update process. This option of the LOGOUT command should be used only as an emergency measure, not as a standard operating practice. See the description of the MASTER command elsewhere in this manual, for a controlled method of forcing a task to logout.

If a user is logged out because the inactivity time has run out, the message TIMEOUT is printed at both the user terminal and the supervisor terminal, followed by the normal logout message. (See the description of the LOUTQM parameter of the system configuration file, in the THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109.) If the user is logged out by the "-user.number" option of the LOGOUT command, or by a LOGOUT command issued at the supervisor terminal, the message FORCED LOGOUT is printed at both the user terminal and the supervisor terminal, followed by the normal logout message.

The command OFF is a synonym for the LOGOUT command within the PRIME INFORMATION environment (but not within the PRIMOS environment).

```
     LOGOUT {-user.number}
```

where "-user.number" is the user number of another user, generally a phantom user, that you wish to logout. The login name of both must be the same. (See the warning above about the use of the "user.number" option.)

FIGURE 1. Format of the LOGOUT command.

```
:LOGOUT

FINNEY (4) LOGGED OUT AT 10'16 032280
TIME USED = 01'24  7'15  1'54
```

User "FINNEY" logged out at 10:16 AM on March 22, 1980 and was connected to the system 1 hour and 24 minutes, used 7 minutes and 15 seconds of CPU time and 1 minute and 54 seconds of disk I/O time.

FIGURE 2. Example of the LOGOUT command.

5.58 LOGTO Command

The LOGTO command allows a user to go from one PRIME INFORMATION account to another without logging out and back in.

A user can move from one account to another using the LOGTO command. The format of the LOGTO command is:

```
     LOGTO account.name {password}
```

This command will attach the user to "account.name". If "account.name" has a password, it must be correctly specified with the LOGTO command.

The LOGTO command does not leave the PERFORM environment, and therefore does not require the user to type "INFO" to begin work with PRIME INFORMATION files and programs. Additionally, the LOGTO command will retain the PERFORM sentence stack described in this manual. Note also that if the new account contains an item named "LOGIN" in its VOC file, that item will be executed by PERFORM before the ":" prompt is issued. (See the section titled "LOGIN Procedure in VOC File" in this manual for more information about this procedure.)

If the LOGTO command is executed for an "account.name" which is not currently a PRIME INFORMATION account, PERFORM displays the message:

```
     This UFD is not a valid INFORMATION account
     Do you wish to make it one? =
```

If the reply to this question is YES, PERFORM will create in the UFD all of the files necessary to make it a PRIME INFORMATION account. If the reply to this question is NO, PERFORM will treat it as a QUIT command (see the section on the QUIT command in this manual), allowing the user to enter the PRIMOS environment in the new account.

```
     LOGTO account.name {password}
```

where:

"account.name" is the new account to be attached to.

"password" is the password of "account.name" (if one has been assigned).

FIGURE 1. Format of the LOGTO command.

```
:LOGTO MURPHY HERO
```

Will attach the user to account "MURPHY" (which has a password of "HERO").

FIGURE 2. Example of the LOGTO command.

5.59 Magnetic Tape Commands

T.ATT, T.REW, T.FWD, T.BCK, T.READ, T.WEOF and T.DET are commands which are used to reserve a tape drive, position and examine a magnetic tape, and write an end-of-file mark and rewind a tape.

T.ATT is used to reserve the mag tape unit for the current user. The attach will fail if another user already has the unit attached. The tape will remain attached to the user until explicitly detatched (T.DET) or until the user logs off the system. Note that T.ATT is a special form of the generalized ASSIGN command, and T.DET is a special form of the generalized UNASSIGN command. Thus, "T.ATT MTU 001" is equivalent to "ASSIGN MT1", while "T.DET" is equivalent to "UNASSIGN MT0". Note also that omission of the unit number as in this "T.DET" example, is the same as specifying the default unit number "MT0".

T.REW will rewind the tape to the load point.

T.FWD will advance the tape until an end-of-file mark is read. After the command has been executed, the tape will be positioned to read the first record after the end-of-file. If the optional record count is included in the sentence (T.FWD rec.cnt), the tape will be advanced past rec.cnt tape records.

T.BCK backs the tape up until an end-of-file mark is read. The tape is then positioned to read the end-of-file mark again. If the optional record count is included in the sentence (T.BCK rec.cnt), the tape will be backed up rec.cnt tape records.

T.READ reads the next tape record and displays it on the screen. This command recognizes the LPTR keyword to print the record on the line printer. If the optional record count is included in the sentence (T.READ rec.cnt), the command will display "rec.cnt" records at a time instead of the default one.

T.WEOF writes an end-of-file mark at the current tape position.

All of these commands recognize the MTU keyword to specify the mag tape mode, tracks and unit number. If the keyword is not used, the commands default MTU to 000 (ASCII, 9-track, unit 0).

```
     T.ATT {MTU mtu}

     T.REW {MTU mtu}

     T.FWD {MTU mtu} {rec.cnt}

     T.BCK {MTU mtu} {rec.cnt}

     T.READ {MTU mtu} {rec.cnt} {LPTR}

     T.WEOF {MTU mtu}

     T.DET {MTU mtu}
```

where:

"m" is mode:

- 0 = no conversions (ASCII assumed data type)
- 1 = EBCDIC conversion
- 2 = invert high bit

"t" is tracks

- 0 = 9 track tape
- 7 = 7 track tape

"u" is unit number

- 0 to 7

"rec.cnt" is the number of tape records

FIGURE 1. Format of the magnetic tape commands.

5.6Ø MAIL Command

The MAIL command is used to send a message to a user on the PRIME INFORMATION system, or to receive mail sent to you by other users.

The MAIL command is provided with PRIME INFORMATION systems to allow users to communicate with one another. Normally, a user may communicate only with the system console, but not with another user (see the section of this manual titled "MESSAGE Command"). The MAIL command allows any user to send "mail" to a specific user by name, or to all users on the system. The format of the MAIL command is:

```
     MAIL {destination}
```

where "destination" may be either a user's login account name, or may be "*" (asterisk) to send mail to all users who login to the system. If the "destination" parameter is omitted, you will be shown first your own private mail (and will be given an opportunity to delete it after you have read it), and then you will be shown mail intended for all users (but not given an opportunity to delete it).

To send private mail to a particular user, type "MAIL account.name", and the MAIL command will prompt you (with a ">" symbol) for each line of your message. A null line (simply NEW LINE, or CARRIAGE RETURN, depending on your terminal) will terminate the message.

To send mail to all users, type "MAIL *", and the MAIL command will prompt you as above for each line of your message.

If someone has sent you mail, you will see a message informing you of this when you LOGIN. To receive this mail, simply type MAIL. If the mail is public (destined for everyone on the system), you will see it, but will not be able to delete it. If the mail is destined for you only, you will be given the option to delete it after you have read it, or to leave it for reading again later.

```
     MAIL {destination}
```

FIGURE 1. Format of the MAIL command.

```
:MAIL PEGGY
>Peggy, let me know when you have finished
>testing the new mailing label program.  We are ready
>to integrate it with the new mass-mailing sales
>letter when you are.
><NEW LINE>
```

FIGURE 2. MAIL command used to send private mail.

```
LOGIN PEGGY

PRIMOS Version 17.2
PEGGY                       (4) LOGGED IN AT 17'13 091080

YOU HAVE PRIVATE MAIL
TYPE 'MAIL' TO READ IT
INFO
PEGGY
Release number 4.2
:MAIL

FROM WAYNE ON 09/09/80 AT 11:55

Peggy, let me know when you have finished
testing the new mailing label program.  We are ready
to integrate it with the new mass-mailing sales
letter when you are.

OK TO DELETE YOUR MAIL FILE? YES
:
```

FIGURE 3. MAIL command used to receive private mail.

5.61 MAKE.MAP.FILE Command

The MAKE.MAP.FILE command is used to create a PRIME INFORMATION data file from the current contents of the system catalog space. After creating this file, the full power of INFORM may be used to produce reports using the information stored in the system catalog.

The catalog space of a PRIME INFORMATION system contains object programs in executable form, with those programs having been placed into the catalog space by using the CATALOG command (described in this manual). When a program is placed into the system catalog space, information relevant to that program is stored with it. Some of the components of that information include catalog name, the date it was cataloged, the account name from which it was cataloged, its size, etc.

This relevant information is directly available by using the MAP command (described in this manual), but the information is presented only in the order in which each program is encountered in the catalog space. In order to apply the full power of INFORM (the high-level on-line query and report generator supplied as part of each PRIME INFORMATION system) to the system catalog, the command MAKE.MAP.FILE is provided, to create an INFORMATION data file from the current catalog contents.

The format of the command is:

```
MAKE.MAP.FILE
```

The file created by this command, for later use by INFORM, is named &MAP&, and is described in more detail in the section of this manual titled "&MAP& File".

The &MAP& file is a standard PRIME INFORMATION data file, described by a dictionary associated with it (D_&MAP&). Once this file is created, reports may be generated from it using the full power of INFORM, including selection criteria and sequencing criteria.

Note that since the &MAP& file is created only upon request, its contents become out-of-date with the first use of any command which manipulates catalog space (such as CATALOG, or DELETE.CATALOG). If you wish to use the &MAP& file to create reports to aid in system catalog management, you should use the MAKE.MAP.FILE command immediately before creating any such reports. The &MAP& file is not updated automatically by any of the catalog management commands, but is simply recreated (at your request, with the MAKE.MAP.FILE command) from current system catalog contents.

```
    MAKE.MAP.FILE
```

FIGURE 1. Format of the MAKE.MAP.FILE command.

```
:MAKE.MAP.FILE
```

FIGURE 2. Example of the MAKE.MAP.FILE command.

5.62 MAP Command

The MAP command shows which INFO/BASIC programs and subroutines are in the catalog space.

The MAP command is used to display the contents of the PRIME INFORMATION System catalog. The format for the MAP command is:

```
MAP
```

The MAP command first displays a summary of catalog space utilization, which includes the percentage of the total catalog space which has been used, and the number of bytes still available for use. Next, the MAP command displays a single line for each program or subroutine in the catalog, with the following information for each entry:

1. The name of the program or subroutine. This is the name which must appear in field two of the verb definition record for programs, or the name by which the subroutine must be CALLed from within an INFO/BASIC program. (See the section titled "VERB Definition Record" in this manual, and the sections titled "SUBROUTINES" and "CALL STATEMENT" in THE INFO/BASIC REFERENCE GUIDE, PDR3903.) The heading of this column is Catalog Name.

2. The DATE when the program or subroutine was cataloged, in the format MMDDYY, where MM is the month, DD is the day, and YY is the last two digits of the year. The heading of this column is Date.

3. The name of the PRIME INFORMATION account from which the program or subroutine was CATALOGed. The heading of this column is Who.

4. The number of arguments required by the program or subroutine when it is invoked. The heading of this column is Args.

5. The number of times the program or subroutine has been invoked for execution since it was initially CATALOGed, or since the CATALOG space was re-organized. The heading of this column is Ref.

6. The size of the executable object code, in bytes. The heading of this column is Object.

7. The size of the symbol cross-reference table, in bytes. The heading of this column is CR.

8. The size of the symbol table, in bytes. The heading of this column is Sym.

9. The total number of bytes in the CATALOGed program or subroutine (i.e., the sum of the object code size, the cross-reference size, and the symbol table size). The heading of this column is Size.

See the description of the MAKE.MAP.FILE command, for a mechanism to use the power of INFORM with the information in the system catalog.

```
    MAP
```

FIGURE 1. Format of the MAP Command.

```
:MAP

Catalog space is   8% full, leaving 1817388 bytes free.
   Catalog Name             Date    Who   Args   Ref  Object   CR   Sym   Size
-WHO                      030780  ISYS     1     0      12    8    52     72
-VCATALOG                 030780  ISYS     2     0     684   80   162    926
-UTILITIES                030780  ISYS     4     0     102   22    76    200
-UNASSIGN                 030780  ISYS     2     0     294   34    94    422
-TWRITE                   030780  ISYS     9     0    1510  166   216   1892
-TREAD                    030780  ISYS    10     0     914  142   202   1258
-TIME                     030780  ISYS     2     0      10   12    62     84
-TERM                     030780  ISYS     2     0     716   80   160    956
-T.WEOF                   030780  ISYS     2     0    1402  126   288   1816
-T.REW                    030780  ISYS     2     0    1308  124   184   1616
-T.READ                   030780  ISYS     2     1    1786  176   276   2238
-T.FWD                    030780  ISYS     2     5    1502  138   338   1978
-T.BCK                    030780  ISYS     2     0    1530  140   332   2002
-SUBSTRINGS               030780  ISYS     4     0      68   34    96    198
-STRS                     030780  ISYS     3     0      68   34    84    186
-SPOOL                    030780  ISYS     2     0     580  106   252    938
-SPLICE                   030780  ISYS     4     0     338   76   110    524
Press <NEW LINE> to continueQ
```

FIGURE 2. Example of the MAP Command.

5.63 MASTER Command

The MASTER command is for use by the System Administrator, to help in controlling the PRIME INFORMATION operating environment.

The MASTER command may be used by the System Administrator to control other terminals on the PRIME INFORMATION system. MASTER allows the System Administrator to release another user's process locks; to enable the break key for another user's terminal; or to force logout a user's task without damaging the data base.

The format of the MASTER command is:

```
     MASTER {action} {option}
```

where "action" is one of the following keywords:

LOCKS	- to manipulate process locks
BREAK	- to manipulate break keys
OFF	- to do a forced logout

and "option" is either the keyword "ALL" (to apply the "action" to all users), or is the specific user number (1-64) to which the "action" should apply.

The MASTER command is a powerful command, whose misuse could create severe problems in a system's database. Therefore, its use is restricted to the System Administrator only. You should not attempt to use the MASTER command unless you understand all of its possible consequences. Its use is protected, so that you will not be able to use it unless you have been taught how to invoke it, or unless your system has been modified.

```
    MASTER {action} {option}
```

FIGURE 1. Format of the MASTER command.

5.64 MENU.DOC Command

The MENU.DOC command is used to produce an expanded listing of a MENU, showing selection text, explanations, and the actual SENTENCE, PARAGRAPH, or MENU that will be executed for each selection.

The MENU.DOC command produces a printed report of the expanded contents of a MENU, being primarily useful for documentation purposes. The format of the MENU.DOC command is:

```
MENU.DOC
```

The MENU.DOC command will prompt for the name of the menu file, and the name of the menu to be expanded and printed. For each menu requested, the MENU.DOC command prints:

1. the descriptive text by which the operator makes a selection from the menu when it is displayed at the terminal;
2. the explanation field which provides additional information about the selection; and
3. the expanded item from the VOC file which would be executed for each selection.

If the ACTION field of a menu is the name of a SENTENCE or PARAGRAPH in the VOC file, then the contents of the VOC file item will be printed in the report produced by MENU.DOC. If the ACTION field is the name of a MENU pointer, then the name of the menu file and the name of the menu to be invoked will be printed.

The MENU.DOC command is normally invoked from within the menu maintenance processor "MENUS", which is supplied with a PRIME INFORMATION system. One of the selections available on the "MENUS" menu is "Print detail of a menu including VOC records referenced", selection number 14. This selection actually invokes MENU.DOC. It would be a good idea to use MENU.DOC on the menu used by "MENUS" (as shown in Figure 2 on the next page), in order to understand the way this command works.

MENU.DOC

FIGURE 1. Format of the MENU.DOC Command.

```
:MENU.DOC
Name of the Menu File=ISYS.MENU.FILE
Name of the Menu=MAKE.MENUS
Name of the Menu=<NEW LINE>
Name of the Menu File=<NEW LINE>
```

The above example will produce a printed listing of the menu used by the menu maintenance processor "MENUS", which resides in the file named "ISYS.MENU.FILE". After the menu is printed, MENU.DOC will prompt again for a new MENU name. A null response (signified by <NEW LINE>) will reissue the "Menu File" prompt again. A null response to this prompt will return you to PERFORM's colon prompt.

FIGURE 2. Example of the MENU.DOC command.

5.65 MENU.PIX Command

The MENU.PIX command is used to print a virtual image of a MENU on the system printer.

The MENU.PIX command prints on the system printer a printed report in the image of a MENU, exactly as it would appear on the terminal screen. This command is intended primarily as an aid in documenting the MENUs created by your installation for your use.

The format of the MENU.PIX command is:

```
     MENU.PIX
```

The MENU.PIX command will prompt for the name of the MENU file, and for the name of the MENU to be printed.

The MENU.PIX command, when used in conjunction with the MENU.DOC command, can be a very powerful documentation tool. The MENU.PIX command produces a printed report which looks exactly like the MENU as it is displayed on the terminal. This report shows the selections available from the MENU, as they would be described to a person using the MENU. The MENU.DOC command produces a printed report showing the actual commands executed for each selection made, and an explanation (if one is available) for each selection on the MENU. See the section titled "MENU.DOC Command" in this manual for more information.

```
MENU.PIX
```

FIGURE 1. Format of the MENU.PIX Command.

```
:MENU.PIX
WHAT IS THE NAME OF YOUR MENU FILE=ISYS.MENU.FILE
WHAT IS THE NAME OF THE MENU PROCDEF=MAKE.MENUS
WHAT IS THE NAME OF THE MENU PROCDEF=<NEW LINE>
WHAT IS THE NAME OF YOUR MENU FILE=<NEW LINE>
```

The above example will produce a printed listing of the virtual image of the MENU maintenance processor "MAKE.MENUS", which resides in the file named "ISYS.MENU.FILE". After the MENU image is printed, MENU.PIX will prompt again for a new MENU name. A null response (signified by <NEW LINE>) will reissue the "MENU FILE" prompt again. A null response to this prompt will return you to PERFORM's colon prompt.

FIGURE 2. Example of the MENU.PIX command.

5.66 MESSAGE Command

The MESSAGE command is used to send a message from your terminal to the system control terminal.

The MESSAGE command is used to send a message to the system control terminal, and results in two lines of printed information at the operator's console. The format of the command is:

```
     MESSAGE text.of.message
```

where "text.of.message" is the message you wish to be sent to the system control terminal. The two lines of information printed at the system control terminal have the following format:

```
     *** uu hh'mm
     text.of.message
```

where "uu" is the user number and "hh'mm" is the time of day in hours and minutes.

```
    MESSAGE text.of.message
```

FIGURE 1. Format of MESSAGE command.

Assume the user types the following:

```
    :MESSAGE PLEASE MOUNT SCRATCH TAPE
```

The following message is printed at the supervisor terminal:

```
    *** 14 17'50
    PLEASE MOUNT SCRATCH TAPE
```

FIGURE 2. Example of the MESSAGE command.

5.67 OFF Command

The last command the user issues to terminate access to the PRIME INFORMATION System is OFF. The OFF command is a synonym for the LOGOUT command within the PRIME INFORMATION environment.

OFF is the last command the user issues when giving up access to the PRIME INFORMATION system. This command is a synonym for the LOGOUT command, described elsewhere in this manual. However, note that OFF is usable interchangeably with LOGOUT only within the PRIME INFORMATION environment, since only PERFORM recognizes these two commands as synonyms. Outside the PRIME INFORMATION environment (i.e., within the PRIMOS command environment), only LOGOUT or its abbreviation LO is recognized by the PRIMOS command processor.

See the section titled "LOGOUT Command" for a complete description of this command, but remember that (unlike LOGOUT), OFF may not be used at the system console.

5.68 P.ATT Command

The P.ATT command may be issued at any user terminal, to request exclusive control of a printer.

A user can request exclusive control of a printer with the P.ATT command. The format of this command is:

```
     P.ATT printer.number {-WAIT}
```

where "printer.number" is an available printer. If printer "printer.number" is currently assigned to another user, the system replies:

```
     DEVICE IN USE
```

unless the optional argument "-WAIT" was supplied. In this case, the P.ATT command is queued until the printer is unassigned by the other user, or until the other user logs out. (See the section titled "P.DET Command" in this manual.)

If any I/O operation is attempted which requires exclusive control of a printer, and that printer has not been assigned to the task with the P.ATT command, the error message:

```
     DEVICE NOT ASSIGNED
```

is displayed at the terminal.

For additional details on the P.ATT command (or ASSIGN command at the PRIMOS level), and on printer unit numbers that can be P.ATTached, see the PRn option of the ASSIGN command in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108.

```
    P.ATT printer.number {-WAIT}
```

FIGURE 1. Format of the P.ATT command.

```
:P.ATT 1            assign printer unit 1

:P.ATT 2 -WAIT      assign printer unit 2,
                    wait for it if necessary
```

FIGURE 2. Examples of the P.ATT command.

5.69 P.DET Command

The P.DET command releases the specified printer, allowing other users to assign it for their use.

The P.DET command may be entered at the user terminal where a printer has been assigned (with the P.ATT command, described elsewhere in this manual). The effect of this command is the opposite of P.ATT. Entered from a user terminal, P.DET unassigns the specified printer, provided that it was previously assigned to the user.

The format of the command is:

```
     P.DET printer.number
```

where "printer.number" is a previously assigned printer. (See the P.ATT command in this manual and the ASSIGN and UNASSIGN commands in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108.)

When a user logs out, all assigned devices (including printers) are unassigned automatically by PERFORM and PRIMOS. The P.DET command is useful for relinquishing control of a printer to other users on the system, but without logging off.

```
    P.DET printer.number
```

FIGURE 1. Format of the P.DET command.

5.70 PASSWD Command

The PASSWD verb allows a user to set or replace an account password.

The PASSWD verb replaces any existing password on the current account with a new password. The format of the command is:

```
PASSWD {owner.password} {non-owner.password}
```

If neither "owner.password" nor "non-owner.password" is specified, both passwords are set to blanks. If only one password is specified, it will become the new owner password, and the non-owner password will be set to blanks.

For additional information concerning account passwords, see the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, and the THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109. The PASSWD command is also a PRIMOS-level command, and is described in both of these references.

```
    PASSWD {owner.password} {non-owner.password}
```

FIGURE 1. Format of the PASSWD command.

```
:PASSWD OMEGA
```

This command sets the owner password to "OMEGA" for the account where the user is currently attached. The non-owner password, since it is omitted from the command, is set to blanks.

FIGURE 2. Example of the PASSWD command.

```
:PASSWD MOZART HAYDN
```

This command sets the owner password to "MOZART" for the account where the user is currently attached. The non-owner password is set to "HAYDN".

FIGURE 3. Another example of the PASSWD command.

5.71 PHANTOM Command

A phantom process is a process which requires no interaction with a terminal. The PHANTOM command will start one of these processes.

A terminal user may initiate a phantom process to perform a job. A phantom process is similar to any other task, but has no terminal associated with it; all controlling input is read from a file rather than from a user terminal. The file contains a sequence of commands and/or user program invocations, and necessary input data specifications, to complete a given job. To start a phantom process, the format is:

```
PHANTOM command.text
```

where "command.text" is usually the name of a paragraph record in the VOC file, but may be the name of a stored sentence, or may be any valid PERFORM sentence which begins with a VERB. The total length of "command.text" may not exceed 25 characters, because PERFORM must concatenate "command.text" with some prefix characters, to create a PRIMOS command file name (which may not exceed 32 characters in length).

When PRIMOS is started, a fixed number of phantom users is specified. (See the NPUSR CONFIG directive, in the THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109.) When a user invokes the PHANTOM command, PRIMOS verifies that there is a free phantom space available. If no free phantom process spaces are available, the message:

```
NO FREE PHANTOMS
```

is printed at the user's terminal. If space is available, the phantom process is logged in and execution begins.

The phantom user feature is useful for running programs which are not interactive and, therefore, do not require the services of a terminal. A process running as a phantom must not perform any terminal I/O; therefore, use the DATA statement in the paragraph for any program requiring input. Any request for input from a terminal causes the phantom process to be logged out. If this condition occurs, the logout message at the supervisor terminal is preceded by the line:

```
PHANTOM TTY REQUEST
```

A user may monitor the status of a phantom process with the STATUS command. A phantom process will have the same login account.name as the user who started it. The STATUS command also prints the user number and line number. The line number is "77" for all phantoms. When the user's PHANTOM job is complete, it is logged out and no longer appears in the output printed by the STATUS command. If a user wishes to stop a phantom that was started at his or her terminal, the command:

```
        LOGOUT -uu
```

must be issued, where "uu" is the task number of the phantom (as reported by the PHANTOM command when it was invoked). A phantom task may also be logged out by use of the LOGOUT command at the supervisor terminal. (WARNING: see the section titled "LOGOUT Command" for possible harmful effects caused by uninformed use of the LOGOUT command.) A user may log out his own task without affecting the status of any phantom tasks started by that user with the PHANTOM command.

For additional information on the files created by the PHANTOM command, see the sections titled "&PH& File" and "PHANTOM Command Files" in this manual.

```
        PHANTOM command.text
```

where "command.text" is the list of commands the phantom is to perform.

FIGURE 1. Format of the PHANTOM command.

```
:PHANTOM EOM.POSTING

Phantom task being performed by User 13

:PHANTOM LIST ORDERS BY DATE LPTR

Phantom task being performed by User 14
```

FIGURE 2. Examples of the PHANTOM command.

5.72 PROP Command

The PROP command is used to display the status of all the SPOOL tasks on the system; to display the characteristics of a particular SPOOL task on the system; and to modify the characteristics of a particular SPOOL task on the system.

PRIME INFORMATION systems normally have at least one SPOOL task active, and may have more than one, to take reports from the SPOOL queue and print them on a printer. Each SPOOL task is associated with one physical printer unit, and each printer unit may be either directly connected to the PRIME INFORMATION system via a parallel printer controller (and therefore located close to the system), or connected via a serial port on the AMLC board (in which case it need not be physically close to the system, and indeed, could be hundreds of miles away).

With the distance between any two printers being potentially great, it becomes important to be able to uniquely identify a particular printer, and to be able to route a particular report to that printer. Additionally, it is important to be able to define the physical characteristics of the form currently in the printer, such as line width and page length. The PRIMOS-level command named PROP is provided to modify and/or display these characteristics; the PRIME INFORMATION-level command, also named PROP, allows access to the capabilities of the PROP command from within PRIME INFORMATION. The format of the PROP command is:

```
     PROP -STATUS

         or

     PROP printer.name -option
```

The PROP command with the "-STATUS" keyword will display the status of all SPOOL tasks defined on the system, indicating whether each one is active (indicated by "started"), or idle (indicated by "stopped").

The PROP command is completely documented in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, and THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109. The "-option" parameter of the second format of the PROP command may be chosen from the list defined in either of these manuals. The only "-option" allowed for the primary system spoolers, "PR0" and "PR1", is "-DISPLAY"; the other "-option" selections result in modification of the characteristics of these system spool tasks, and this modification is restricted to the System Administrator using the operator's console.

The PROP command allows a user to create a spooler environment, which can then be controlled with the "-MODIFY", "-START", "-STOP", or "-DELETE" options. This is particularly useful for remote AMLC spoolers, where a user is more likely to need to control the physical printer (for example, to put a different form into the printer). The ability to modify an environment is allowed only if the user's LOGIN name matches the LOGIN name of the user who created that spooler environment.

```
     PROP -STATUS

         or

     PROP printer.name -option
```

FIGURE 1. Format of the PROP command.

```
:PROP -STATUS
[PROP rev 17.2]

PRØ started
PR1 stopped
ACCTNG started
PAYROL stopped

:PROP PRØ -DISPLAY
[PROP rev 17.2]

DEVICE: PRØ
PAPER: 'CHECKS'
MESSAGE: System will be down tonight after 5:ØØ.

COMOUT: OFF
UPCASE: OFF
PRINT: ON
PLOT: OFF
LENGTH:    38
LARGE:     2Ø
LIMIT: 3ØØØØ
UPPER:     63
LOWER:      Ø
HEADER:     1
WIDTH:     8Ø
LINES:      Ø
```

FIGURE 2. Example of the PROP command.

5.73 PSPOOL Command

The PSPOOL command is a PRIME INFORMATION-accessible version of the PRIMOS-level SPOOL command. Its principal use is to support the PRIME INFORMATION-level SPOOL command, and it is not normally used as a command entered at a terminal.

PRIME INFORMATION systems have two similar commands in the VOC file: SPOOL, and PSPOOL. The SPOOL command (described elsewhere in this manual) is the one normally used, while the PSPOOL command is used by the SPOOL command (and therefore must have an entry in the VOC file which describes it). The PSPOOL command is documented here for explanation only. The format of the PSPOOL command is:

```
PSPOOL treename {-options}
```

where "treename" is a complete description of the file path that uniquely describes the SAM file to be spooled, and "-options" is an optional list of parameters (described in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108).

The PSPOOL command is exactly the same as the PRIMOS-level SPOOL command, as documented in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108. When used directly from within PRIME INFORMATION, the "treename" parameter must specify a record in a Type 1 file, or a PRIMOS-level SAM (Sequential Access Method) file which contains textual data; a COMO file is an example of a SAM file.

Note that the PSPOOL command ignores the printer characteristics defined by the SETPTR command. Remember that PSPOOL is a PRIMOS-level command, and is not aware of any characteristics of the PRIME INFORMATION environment. Unless you understand well the distinctions between PRIMOS and PRIME INFORMATION, you should not use the PSPOOL command, but instead should use its "twin", SPOOL.

```
     PSPOOL treename -options
```

FIGURE 1. Format of the PSPOOL command.

```
:PSPOOL DEMO>BP>PROGRAM1 -COPIES 2
[SPOOL rev 17.2]
Your spool file, PRT008, is 11 records long.
:
```

FIGURE 2. Example of the PSPOOL command.

5.74 PTERM Command

The PTERM command allows the user to set or display some of the video display terminal characteristics. The PTERM command is a PRIME INFORMATION-level version of the PRIMOS-level TERM command.

The general form of the PTERM command is:

```
    PTERM option-1 ... option-n
```

where "option-1" through "option-n" may be chosen from the list defined in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108. Briefly, that list is summarized here (and the list is displayed on your terminal in response to the PTERM command with no "option"):

```
    -BREAK {ON/OFF}
    -DISPLAY
    -ERASE character
    -FULL
    -HALF {-LF/-NOLF}
    -KILL character
    -NOXOFF
    -XOFF
```

One additional parameter has been added to the PRIMOS TERM command for the use of PRIME INFORMATION (via the PTERM command): the parameter -ECHO with the keywords DELAY or NODELAY, which is used by PRIME INFORMATION's terminal handler to determine when data typed by the terminal operator will be echoed to the terminal display. If you type "PTERM -ECHO DELAY", then all subsequent typing that you do will not be echoed back to your terminal display until the appropriate time. For example, if you are "typing ahead" of requests for input, then the first data that you type will be echoed (displayed) immediately, but the remainder of what you type will not be echoed until the actual INPUT statement is executed requesting the next data. Other noticeable effects of the -ECHO DELAY option include that the backspace function will erase characters as you backspace over them, and the KILL function (CONTROL-X, unless modified by your System Administrator) will return the cursor to the beginning of the line, after erasing all typed data. When the program executing the INPUT statement is using cursor positioning, the -ECHO DELAY option will not allow you to backspace beyond the beginning of the current cursor position set by the program.

To reset normal terminal echo characteristics, type "PTERM -ECHO NODELAY". This is the default mode on PRIME INFORMATION systems. To set other characteristics of the display terminal, see the DELAY, HUSH and TERM commands in this manual, and the AMLC command in the THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109.

```
     PTERM option-1 ... option-n
```

The options are:

```
     -BREAK {ON/OFF}
     -DISPLAY
     -ECHO {DELAY/NODELAY}
     -ERASE character
     -FULL
     -HALF {-LF/-NOLF}
     -KILL character
     -NOXOFF
     -XOFF
```

FIGURE 1. Format of the PTERM command.

```
:PTERM -DISPLAY
ERASE = IN OCTAL = 000210
 KILL  =  IN OCTAL = 000230
FULL DUPLEX
X-OFF/X-ON ENABLED

:PTERM -ECHO DELAY

:PTERM -DISPLAY
ERASE = IN OCTAL = 000210
 KILL  =  IN OCTAL = 000230
HALF DUPLEX
NO LINE FEED
X-OFF/X-ON ENABLED
ECHO DELAY
```

FIGURE 2. Examples of the PTERM command.

5.75 PTIME Command

The PTIME command prints the current values stored in the PRIMOS time accounting registers, which are: total connect (login) time; total CPU time used; and total I/O time.

The PTIME command prints the current value stored in the PRIMOS time accounting registers, which are: total connect (login) time; total CPU time used; and total I/O time. The format of this command is:

```
PTIME
```

The three values printed are the same as the three values in the logout message:

Connect Time	(hours, minutes)	Time since LOGIN.
Compute Time	(minutes, seconds)	Time accumulated executing commands or using programs (does not include disk I/O time).
Disk I/O Time	(minutes, seconds)	Time accumulated for disk input/output.

The disk I/O (input/output) time includes user-requested I/O to files, and also paging I/O time generated on the user's behalf. All times include supervisor services, such as the time spent executing supervisor subroutines on the user's behalf. Some supervisor service associated with the PRIMOS scheduler is charged to the supervisor (at the system control terminal) and not to the user. When the system is idle, CPU time is charged to the supervisor task.

The PTIME command is a PRIME INFORMATION-level version of the PRIMOS TIME command, which is described in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108.

For a display of system time of day instead of cumulative task time, see the description of the TIME command elsewhere in this manual.

```
     PTIME
```

FIGURE 1. Format of the PTIME command.

```
:PTIME

 2'24  3'10  1'34
```

This user has been logged in for 2 hours and 24 minutes, during which time he/she has used 3 minutes and 10 seconds of CPU time, and 1 minute and 34 seconds of disk I/O time.

FIGURE 2. Example of the PTIME command.

5.76 QUIT Command

When the user wishes to leave the PRIME INFORMATION environment and enter the PRIMOS environment, this may be done by the QUIT command.

QUIT is the last command the user issues when the intention is to give up access to the PRIME INFORMATION system, but retain access to the PRIMOS-level portion of the system. The format of this command is:

```
     QUIT
```

When this command is issued, all PRIME INFORMATION files are closed and the user enters the PRIMOS environment (signalled by the PRIMOS prompt, "OK, "). Additionally, the PERFORM sentence stack is erased (if the item named STACKWRITE in the user's VOC file has "OFF" in field two), or is preserved until the next session (if the item named STACKWRITE in the user's VOC file has "ON" in field two). (See the section titled "PERFORM Sentence Stack Retention (STACKWRITE)" in this manual for more information on the preservation of the sentence stack.)

The QUIT command should not be confused with the LOGOUT or OFF commands. The QUIT command leaves the terminal logically connected to the system (i.e., still logged in), but no longer able to access PRIME INFORMATION files or programs. Instead, the terminal may now access PRIMOS files, via programs written in FORTRAN, COBOL, RPG, etc. If the user subsequently desires to return to the PRIME INFORMATION environment, it is only necessary to type "INFO" in response to the "OK, " prompt of PRIMOS.

The LOGOUT and OFF commands are used to completely relinquish access to the PRIME INFORMATION system, whether at the PRIME INFORMATION level (for either the LOGOUT or the OFF command), or at the PRIMOS level (for the LOGOUT command only). After issuing either the LOGOUT or OFF commands, it is necessary to LOGIN again to regain access to the system.

```
    QUIT
```

FIGURE 1. Format of the QUIT command.

```
:QUIT

OK,
```

FIGURE 2. Example of the QUIT command.

5.77 RADIX, XTD, DTX, BTD, DTB, OTD and DTO Commands

The RADIX commands will convert from one number base to another.

The RADIX commands may be used to convert from hexadecimal, binary, or octal to decimal; and from decimal to hexadecimal, binary, or octal. The format of the RADIX commands are:

RADIX	Prompt for conversion type.
XTD {hexadecimal string}	Hexadecimal to decimal conversion.
BTD {binary string}	Binary to decimal conversion.
OTD {octal string}	Octal to decimal conversion.
DTX {decimal string}	Decimal to hexadecimal conversion.
DTB {decimal string}	Decimal to binary conversion.
DTO {decimal string}	Decimal to octal conversion.

The first form of the command, "RADIX", prompts the user for the specific conversion desired (XTD, DTX, etc.), and then prompts for the string to be converted.

The remaining forms of the number base conversion commands may be used with or without the string to be converted. If the string to be converted is typed with the command, the string will be converted and control will be returned to PERFORM. If the optional string is omitted when the command is typed, the command processor will prompt for a string to be converted or for a new action (DTB, DTX, etc.), converting each new entry and prompting for another one, until a <NEW LINE> with no data is the response to the prompt; then control will return to PERFORM.

```
    RADIX

    XTD {hexadecimal string}        Hexadecimal to decimal conversion

    BTD {binary string}             Binary to decimal conversion

    OTD {octal string}              Octal to decimal conversion

    DTX {decimal string}            Decimal to hexadecimal conversion

    DTB {decimal string}            Decimal to binary conversion

    DTO {decimal string}            Decimal to octal conversion
```

FIGURE 1. Format of the RADIX command.

```
:RADIX
Input Action Code - XTD, DTX, BTD, DTB, OTD, DTO =DTX
Input Decimal string or new Action Code =255
Hex  FF
Input Decimal string or new Action Code =<NEW LINE>

:XTD FD
Decimal  253

:DTB
Input Decimal string or new Action Code =22
Binary  10110
Input Decimal string or new Action Code =BTD
Input Binary string or new Action Code =10110
Decimal  22
Input Binary string or new Action Code =<NEW LINE>

:OTD 11
Decimal   9
```

FIGURE 2. Examples of the RADIX commands.

5.78 RECORD Command

The RECORD command determines which group a record should be placed into within a particular file, and whether the record presently exists in the group. It is used principally for verification of file integrity.

The RECORD command instructs INFO/DMS to do a trial hash-encoding of a specified record identifier into an existing PRIME INFORMATION data file. The format of this command is:

```
     RECORD {file.name} {record.id}
```

The RECORD command displays the group number that a record with the identifier "record.id" would be placed into within the file "file.name". This command will additionally display whether the record identified by "record.id" exists in that group within file "file.name". If the RECORD processor reports that "record.id" was not found in this group, but you believe that it should have been, this might be an indication of a damaged file. You should report this to your System Administrator for corrective action.

```
RECORD {file.name} {record.id}
```

FIGURE 1. Format of the RECORD command.

```
:RECORD STATES MA
"MA" hashes to group 2 and was found.
```

FIGURE 2. Example of the RECORD command.

5.79 RELEASE Command

Records in PRIME INFORMATION files which have been locked by the INFO/BASIC READU, MATREADU, and READVU statements can be unlocked by the RELEASE command. This command uses a SELECT list if one is active.

The RELEASE command allows a user to release record locks set by READU, MATREADU, and READVU statements of INFO/BASIC programs. The format of the RELEASE command is:

```
     RELEASE {file.name} {record.id}...
```

Each record lock is identified by user number (terminal number), an internal file identifier, and an abbreviated record id. The RELEASE command will release all record locks for the terminal number of the user who set the locks. If "file.name" is specified but not "record.id", all record locks for "file.name" are released. If both "file.name" and "record.id" are omitted, all the record locks for the user are released.

The importance of the terminal number deserves additional emphasis. If you were logged in to account "PAYROLL" using terminal number 3 and the INFO/BASIC programs you used locked some records, the RELEASE command would unlock those records for you only if you are still logged in on terminal number 3. If you move to terminal number 4 and log in to account "PAYROLL", RELEASE will not release locks set by your task while you were on terminal number 3.

Note that the use of this command should occur very infrequently. Normal termination of INFO/BASIC programs will cause record locks set by the READU, MATREADU, and READVU statements to be released. The most likely use of this command might be to recover from an error, such as a terminal ceasing to function while records were locked.

A similar command that releases all record locks in a named file is the RELEASE.ITEMS command, documented in this manual. Also, to display record locks currently active, see the LIST.READU command description in this manual.

```
    RELEASE {file.name} {record.id}...
```

FIGURE 1. Format of the RELEASE command.

```
:RELEASE STATES AK
1 record(s) released.

:RELEASE STATES MN NH VT KY
4 record(s) released.

:SELECT STATES 'TX' 'AL' 'CO' 'KS'

4 records selected

:RELEASE STATES
4 record(s) released.
```

FIGURE 2. Examples of the RELEASE command.

5.80 RELEASE.ITEMS Command

Records in PRIME INFORMATION files which have been locked by the INFO/BASIC READU, MATREADU, and READVU statements can be unlocked by the RELEASE.ITEMS command.

The RELEASE.ITEMS command allows a user to release record locks set by READU, MATREADU, and READVU statements of INFO/BASIC programs. The format of the RELEASE.ITEMS command is:

```
     RELEASE.ITEMS
```

Each record lock is identified by user number (terminal number), an internal file identifier, and an abbreviated record id. The RELEASE.ITEMS command will release all record locks for the terminal number of the user who set the locks.

The importance of the terminal number deserves additional emphasis. If you were logged in to account "PAYROLL" using terminal number 3 and the INFO/BASIC programs you used locked some records, the RELEASE.ITEMS command would unlock those records for you only if you are still logged in on terminal number 3. If you move to terminal number 4 and log in to account "PAYROLL", RELEASE.ITEMS will not release locks set by your task while you were on terminal number 3.

Note that the use of this command should occur very infrequently. Normal termination of INFO/BASIC programs will cause record locks set by the READU, MATREADU, and READVU statements to be released. The most likely use of this command might be to recover from an error, such as a terminal ceasing to function while records were locked.

To release record locks for individual files or records see the RELEASE command. Also, to display record locks currently active, see the LIST.READU statement.

```
    RELEASE.ITEMS
```

FIGURE 1. Format of the RELEASE.ITEMS command.

```
:RELEASE.ITEMS
:
```

FIGURE 2. Example of the RELEASE.ITEMS command.

5.81 RESET.PRINTER Command

The RESET.PRINTER command allows a programmer to change printer characteristics from within an INFO/BASIC program. It is intended that its use be via an EXECUTE statement, rather than being invoked directly from a terminal.

The RESET.PRINTER command is provided to allow alteration of printer characteristics at a time other than is normally allowed by PRIME INFORMATION. Normally, printer characteristics may be changed immediately at PERFORM level, using the SETPTR command (described elsewhere in this manual). Note that this change becomes effective only when at PERFORM's colon prompt, except by use of RESET.PRINTER. While it is possible to "EXECUTE" the SETPTR command from within INFO/BASIC, it is necessary to additionally "EXECUTE" the RESET.PRINTER command in order to over-ride PERFORM's protection against unintended modification of printer characteristics. This unintended modification would happen, for example, if an INFO/BASIC program used the EXECUTE statement to invoke a paragraph that contained the SETPTR command, after the INFO/BASIC program had just executed its own SETPTR command.

At the moment PRIME INFORMATION leaves the colon prompt (when a program is beginning execution), the printer characteristics which have been set before invoking the program are "frozen". These printer characteristics may be modified by the INFO/BASIC program (via an "EXECUTE SETPTR" statement) only once, and then only before the first "PRINT" statement has been executed. Any "EXECUTE SETPTR" statement encountered after the first "PRINT" statement, or after any other "EXECUTE SETPTR" statement, will not affect the printer characteristics, <u>unless</u> an "EXECUTE RESET.PRINTER" statement is used before it. The execution of RESET.PRINTER "unfreezes" printer characteristics and allows the user to change them from within an INFO/BASIC program.

As an example, it would be necessary to use RESET.PRINTER if an INFO/BASIC program needed to establish its own printer characteristics (via an EXECUTE of a SETPTR command), then needed to establish a different set of printer characteristics later in the program. The later SETPTR parameters would not be allowed to override the existing ones until the program returns to PERFORM's colon prompt, unless RESET.PRINTER is executed before trying to establish new printer characteristics.

The format of the command is:

```
     EXECUTE 'RESET.PRINTER'
     EXECUTE 'RESET.PRINTER'
```

FIGURE 1. Format of the RESET.PRINTER command.

```
          PRINT 'START OF PROGRAM'
     *    Print salesorders using existing 'SETPTR' characteristics
          EXECUTE 'LIST SALESORDERS LPTR'
     *    Tell PERFORM we want to reset 'SETPTR' characteristics
          EXECUTE 'RESET.PRINTER'
     *    Now set new printer characteristics, for printing checks
          EXECUTE 'SETPTR ,,,,,,FORM CHECKS,BRIEF'
     *    Ready to print checks on proper forms named 'CHECKS'
          EXECUTE 'LIST CHECK.FILE LPTR'
          PRINT 'END OF PROGRAM'
     END
```

FIGURE 2. Example of the RESET.PRINTER command.

5.82 RESIZE Command

The RESIZE command is used to re-organize an existing PRIME INFORMATION file using a new, user-specified file type and modulo, to retain maximum efficiency in the storage and retrieval of records from the file.

Maximum efficiency in PRIME INFORMATION data file storage and retrieval is achieved when the records in the file are evenly distributed among the groups which comprise the file. When an examination of the file (by GROUP.STAT, GROUP.STAT.DETAIL, HASH.TEST, or HASH.TEST.DETAIL commands, described elsewhere in this manual) indicates the need to re-optimize the distribution of records in the file, this may be easily accomplished with the RESIZE command. The RESIZE command will physically re-organize an existing PRIME INFORMATION file, utilizing a new user-supplied file type and/or modulo, without the need for copying the file, re-allocating it, etc. RESIZE also deletes any physical disk records in a group that are no longer needed due to re-organization of the group. The format of the RESIZE command is:

```
     RESIZE {DICT} {file.name} {type} {modulo}
```

The "DICT" and "file.name" parameters may be omitted as a signal for this verb to use a SELECT list. The "type" and "modulo" parameters are the same as for the CREATE.FILE command, except that Type 1 is not acceptable for the RESIZE command. An asterisk (*) for either of these parameters is a signal to use the existing type or modulo, respectively.

Essentially, RESIZE will re-organize the PRIME INFORMATION file with the new file type and modulo as specified, and will then re-distribute all of the records from the original file groups into their appropriate groups in the new file. When finished, the PRIME INFORMATION file will appear as though it had been allocated originally with the file type/modulo combination used as parameters to the RESIZE command. The equivalent function could be performed by: creating a temporary file with the new file type/modulo; copying all records from the original file to the temporary file (and from the dictionary of the original file to the dictionary of the temporary file); deleting the original file; and finally, changing the name of the temporary file to that of the original file.

Before you utilize the RESIZE command, you may wish to try several combinations of file type and modulo without actually affecting the file. See the descriptions of the HASH.TEST and HASH.TEST.DETAIL commands in this manual for how to do this. Also see the sections titled "HASH.HELP Command" and "HASH.HELP.DETAIL Command", and "PRIME INFORMATION File Groups" in this manual.

```
     RESIZE {DICT} {file.name} {type} {modulo}
```

FIGURE 1. Format of the RESIZE command.

```
:GROUP.STAT STATES
 Type description= Hashed, keys end in letters
 Bytes  Records             File= STATES    Modulo= 1    Type= 4
  6458     51 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
======  ====
  6458     51  Totals
  6458     51  Averages per group
     0      0  Standard deviation from average
   0.0    0.0  Percent std dev from average
:RESIZE STATES 3 5
STATES
:GROUP.STAT STATES
 Type description= Hashed, keys end mainly in numbers
 Bytes  Records             File= STATES    Modulo= 5    Type= 3
  1021      8 >>>>>>>>
  1818     14 >>>>>>>>>>>>>>
   274      2 >>
  1121      9 >>>>>>>>>
  2248     18 >>>>>>>>>>>>>>>>>>
======  ====
  6482     51  Totals
  1296     10  Averages per group
   682      5  Standard deviation from average
  52.6   50.0  Percent std dev from average
```

FIGURE 2. Example of the RESIZE command.

5.83 RUN Command

INFO/BASIC programs that have been compiled may be executed by use of the RUN command.

The RUN command has the following form:

```
     RUN {file.name} program.name {NOPAGE}
```

Object code for the program "program.name" is generated by compiling the INFO/BASIC program using the BASIC command. The RUN command loads this object code and executes the program.

Note that if "file.name" is not specified, a file name of "BP" is assumed. Note also that, although the object code produced by the INFO/BASIC compiler is given the name "$program.name", the RUN command parameter "program.name" is specified without the "$" prefix. This "$" prefix is supplied automatically by the RUN command.

The optional NOPAGE parameter will eliminate the default paging mode maintained by PERFORM. For programs which are not using cursor positioning (the "@(column.number,line.number)" function), PERFORM counts lines printed by the PRINT statement, and displays "Press <NEW LINE> to continue ..." at the bottom of each full page of display. If you use the NOPAGE option with the RUN command, this pagination will no longer be done by PERFORM. See the section of THE INFO/BASIC REFERENCE GUIDE, PDR3903 titled "AT-SIGN (@) FUNCTION" for more information on pagination.

```
    RUN {file.name} program.name {NOPAGE}
```

If "file.name" is not specified then "BP" is assumed.

FIGURE 1. Format of the RUN command.

```
:RUN PRODUCTION RPT.PGM1
```

This command will run a program named "RPT.PGM1" from a file named "PRODUCTION".

FIGURE 2. Example of the RUN command.

5.84 SAVE.LIST Command

Lists of record identifiers that have been constructed by the SELECT and SSELECT commands may be stored for subsequent reference by name. These select lists are stored for future processing by the SAVE.LIST command.

After contructing a list of record identifiers which meet a specified set of selection criteria (with the verbs SELECT or SSELECT, described in THE INFORM REFERENCE GUIDE, IDR3905), this list may be stored for subsequent retrieval by name. Once stored, a select list may be retrieved by its name (by the GET.LIST command, described elsewhere in this manual) to prepare for processing, as if the select list had just been created by the SELECT or SSELECT verb. This technique has the advantage of allowing use of a user-specified select list more than once, without having to re-access the data file to recreate the select list.

Immediately after a select list has been created (with the SELECT or SSELECT commands), it may be stored in the file named &SAVEDLISTS&, using the following command:

```
    SAVE.LIST {list.name}
```

If the "list.name" parameter is specified, the select list will be saved in the &SAVEDLISTS& file using that name. If the "list.name" parameter is not specified, the select list will be saved under the name &TEMPport#&, where "port#" is the user's port number. In either case, the SAVE.LIST command automatically appends a three-digit sequence number to the select list name. This sequence number is 000 for select lists which contain fewer than 5000 record identifiers.

For saved select lists which contain more than 5000 record identifiers, the SAVE.LIST command creates multiple records in the &SAVEDLISTS& file, each with the name "list.name" or "&TEMPport#&", as appropriate, and with sequence numbers from 001 to nnn. The user need never be concerned with these sequence numbers, or with how many records were needed to accomplish the SAVE.LIST command, since this is managed automatically by the three commands which work with saved lists (SAVE.LIST, GET.LIST, and DELETE.LIST). However, the user must be aware of these sequence numbers when using the PRIME INFORMATION EDITOR to examine or modify saved lists, since the EDITOR does not append sequence numbers to the specified list name.

See also the section in this manual titled "&SAVEDLISTS& File".

```
SAVE.LIST {list.name}
```

FIGURE 1. Format of the SAVE.LIST Command.

```
:SELECT STATES WITH SETTLED > 1699 AND SETTLED < 1800

 18 records selected

:SAVE.LIST STATES.1700
```

FIGURE 2. Example of the SAVE.LIST Command.

```
:SORT &SAVEDLISTS&

SORT &SAVEDLISTS& 10:25:47  04-03-80  PAGE      1
&SAVEDLISTS&....................

&TEMP2&001
&TEMP2&002
&TEMP2&003
&TEMP3&000
&TEMP7&000
BACK.ORDERS000
COMMANDS000
FILES000
STATES.1700000

9 records listed.
```

FIGURE 3. Records in the &SAVEDLISTS& File.

5.85 SELECT (INFORM System Processor)

SELECT is a portion of the high-level, natural language QUERY and REPORT GENERATOR portion of the PRIME INFORMATION system. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. SELECT is one of these INFORM processors.

5.86 SETFILE Command

The SETFILE command allows the user to create a file pointer (or synonym file name) to an existing PRIME INFORMATION file.

The SETFILE command is used to create a file definition record in the VOC file that points to a PRIME INFORMATION file which already exists on the system. The file pointed to may be a file in a different account, or a file with a different name in the same account. The format of the SETFILE command is:

```
     SETFILE {treename} {file.name} {OVERWRITING}
```

where "treename" is the PRIMOS treename which uniquely identifies the file being pointed to, and "file.name" is the name by which the file is to be accessed from this account.

The OVERWRITING keyword, if used, will cause any existing entry in the VOC file with the same name as "file.name" to be replaced with this new entry. Note that the SETFILE command does not check to see if it is replacing a file pointer or synonym when the OVERWRITING keyword is specified; this keyword should be used with caution and knowledge of the contents of the VOC file.

As with most PERFORM verbs, the SETFILE verb will prompt for omitted parameters, or will accept them all on the same line with the command. However, if the account which contains the file requires a password, it is not possible to use the single-line format, since the "treename" and the password would be treated as the "treename" and "file.name" by PERFORM. An example of SETFILE for a "treename" which requires a password follows:

```
:SETFILE
Enter tree name =PAYROLL SECRET>SALARY
What do you want to call it in your VOC File =SALARY

Establish file pointer
Treename      PAYROLL SECRET>SALARY
VOC Name      SALARY

OK to establish pointer (Y/N)  =Y
```

```
SETFILE {treename} {file.name} {OVERWRITING}
```

FIGURE 1. Format of the SETFILE command.

```
:SETFILE DEMO>STATES STATES OVERWRITING


:SETFILE
Enter tree name =DEMO>STATES
What do you want to call it in your VOC File =MY.STATES

Establish file pointer
Treename      DEMO>STATES
VOC Name      MY.STATES

OK to establish pointer (Y/N)  =Y


:SETFILE
Enter tree name =DEMO>STATES
What do you want to call it in your VOC File =STATES

Establish file pointer
Treename      DEMO>STATES
VOC Name      STATES

OK to establish pointer (Y/N)  =Y
Pointer STATES already exists, do you want to overwrite (Y/N )=N
What do you want to call it in your VOC File =NEW.STATES

Establish file pointer
Treename      DEMO>STATES
VOC Name      NEW.STATES

OK to establish pointer (Y/N)  =N
SETFILE aborted.
:
```

FIGURE 2. Examples of the SETFILE command.

5.87 SETPTR Command

The line printer spooler options for each of the 256 logical print units are set with the SETPTR command.

The SETPTR command is used to set the spooler options for logical print units 0 thru 255. These options remain in effect until explicitly changed (by using the SETPTR command again), or until you leave the PRIME INFORMATION System (LOGOUT), or leave the PRIME INFORMATION environment (QUIT). The format of the SETPTR command is:

```
SETPTR unit, l.len, p.len, t.marg, b.marg, mode {,option}...
```

where:

unit	logical print unit, 0 thru 255 (default 0)	
l.len	line length, or paper width (default 132)	
p.len	page length, or lines per page (default 60)	
t.marg	top margin in number of lines (default 3)	
b.marg	bottom margin in number of lines (default 3)	
mode	1	output to the line printer spooler (default).
	2	output to an assigned AMLC line (see below).
	3	output to a hold file (see below).
	4	output to an assigned tape (see below).
	5	output to an assigned printer.

The options are:

BANNER	name	Specifies "name" is to be the second line on banner page; first line is the account name. (if Mode 3, see below.)
BRIEF		Do not ask the user if the parameters are o.k. (useful in SENTENCES and PARAGRAPHS).
DEFER	time	Defer printing until the specified "time" (in either 24-hour format, or 12-hour format).
EJECT		Do a page eject at the end of the report.
FORM	name	Specifies a special form "name". The file is not printed until the system operator indicates that form "name" is in the printer; "name" must be 6 characters or less.

FTN		Use FORTRAN control conventions (described in the PRIMOS SUBROUTINES REFERENCE GUIDE, PDR3621).
HOME		Print on home printer (if on a network).
LNUM		Prefix each line with a line number.
NFMT		The spooler should supply no format control; the user is explicitly controlling paging.
NHEAD		Do not print a banner.
PLOT	nwords	Denotes a plot file; "nwords" is the decimal number of words to be output per raster scan.

Mode 2: To an AMLC line (for a remote serial printer).

If Mode 2 is specified, all printer output will go directly to an assigned AMLC line. SETPTR Mode 2 must be preceded by "ASSIGN AMLC n", where "n" represents a terminal port (see the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, and the section titled "ASSIGN Command" in this manual); the SETPTR "unit" parameter must be the AMLC line number that has been previously assigned. Subsequent print commands to the printer unit specified in the SETPTR command will be directed to the assigned AMLC line.

Mode 3: HOLD file.

If Mode 3 is specified in a SETPTR command, all printer output will be directed to a file named "&HOLD&". If the file, "&HOLD&", does not already exist in the account, SETPTR will create "&HOLD&" and its dictionary, "D_&HOLD&". The "&HOLD&" file may not be a remote file.

To allow for successive reports to "&HOLD&", the following naming conventions are used:

Option 1:	If no "BANNER" name is specified, the output report is named P#0000; subsequent reports will overlay P#0000.
Option 2:	If "BANNER UNIQUE" is specified, then the output report will be named P#0000_nnnn, where "nnnn" is a counter that is automatically incremented for each SETPTR command.
Option 3:	If "BANNER report.name" is specified, then the output report will be named "report.name".
Option 4:	If "BANNER UNIQUE report.name" is specified, then the output report will be named "report.name_nnnn", where "nnnn" is a counter. "UNIQUE" must be the first argument and "report.name" the second argument for this option.

The counter used by Option 2 and Option 4 is unique to each "&HOLD&" file, and is stored in the record named "NEXT.HOLD" in the dictionary "D_&HOLD&". The counter is set to 1 with the initial creation of the "&HOLD&" and "D_&HOLD&" files, and is incremented by 1, to 9999 before being reset to 1. The counter is incremented at the execution of each SETPTR command, not as each file is printed. The user must execute a SETPTR using Mode 3 before each report, to get unique report names in "&HOLD&". When a printed report is produced, if an output report record with the same name exists in "&HOLD&", the original report will be deleted and overwritten.

Mode 4: Direct to TAPE.

If Mode 4 is used, all output destined for the line printer will be written to tape in report format; the output can later be spooled directly from tape to a printer by using the SP.TAPE command (described elsewhere in this manual). Mode 4 might be used to produce a printed report for archival purposes only, without actually printing it; because available disk space is low; or if a large report is being transferred to another system for printing.

If the following form of the SETPTR command is used:

```
SETPTR unit
```

all of the current parameters for printer "unit" are displayed.

```
:SETPTR 0
Width       132
Length      60
Top margin 3
Bot margin 3
Mode        1

2048
Options are :

:SETPTR 0,80,66,3,3,1,FORM WHITE
Width       80
Length      66
Top margin 3
Bot margin 3
Mode        1

2048
Options are :
Form WHITE;
OK to set parameters as displayed ? (enter <RETURN> or "N")<RETURN>

:SETPTR 3,120,55,3,3,1,BRIEF
```

FIGURE 1. Examples of the SETPTR command.

5.88 SLEEP Command

The SLEEP command allows a process to be suspended for a specified period of time, or until a specified time of day.

The SLEEP command has two forms:

```
     SLEEP {nsecs}

         or

     SLEEP hh:mm{:ss}
```

where "nsecs" is the number of seconds a process must wait before beginning, and "hh:mm:ss" is the time in hours, minutes, and (optionally) seconds, in 24-hour format, when a process is to be started. If neither parameter is entered, the default value of 1 second is used.

The SLEEP command is used most often in paragraphs, for phantom processes which are to be invoked earlier than their actual intended beginning time. This use allows convenience in starting a process that should run late at night, after all interactive users have gone home, since the phantom process can be invoked at a convenient time and simply have its actual beginning delayed. (See the "PHANTOM Command" description elsewhere in this manual.)

Note that, since this command uses 24-hour format, you could specify a time of day that has already passed today. In this event, the task will not begin until the next day at that time. An example:

```
     :TIME
     11:52:23  3 NOV 1980
     :SLEEP 10:30
```

In this example, if the actual intent was to have the task delay until 10:30 the next morning, then the SLEEP command was used correctly. However, if the intent was to have the task resume execution at 10:30 in the evening, the command should have been:

```
     :SLEEP 23:39
```

```
    SLEEP {nsecs}

        or

    SLEEP hh:mm{:ss}
```

where:

"nsecs" is the number of seconds to sleep

"hh:mm:ss" is the time in hours, minutes, and seconds, with seconds optional, to begin (24-hour format).

(a default of 1 second is used if no parameter is used.)

FIGURE 1. Format of the SLEEP command.

```
:SLEEP 9Ø      Suspend execution for 1 minute 3Ø seconds.

:SLEEP 15:28   Suspend execution until 3:28 in the afternoon.

:SLEEP         Suspend execution for 1 second.
```

FIGURE 2. Examples of the SLEEP command.

5.89 SORT (INFORM System Processor)

SORT is a portion of the high-level, natural language QUERY and REPORT GENERATOR portion of the PRIME INFORMATION system. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905. This command uses a SELECT list if one is active.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. SORT is one of these INFORM processors.

5.90 SP.TAPE Command

The SP.TAPE command is used to read a SPOOL tape which was created by using the SETPTR command's Mode 4, and to print its contents directly on a system printer.

One of the options of the SETPTR command (described elsewhere in this manual) directs that records which would ordinarily go to the system SPOOL queue, instead should be routed to a tape; this option is selected by specifying Mode 4 when the SETPTR command is invoked. This option might be used to conserve disk space (since a very large report would take a large amount of SPOOL queue space), or it might be used to archive a report without ever printing it. When the report that was placed on tape needs to be printed, the SP.TAPE command is provided for that purpose.

The format of the SP.TAPE command is:

```
     SP.TAPE {printer.number} {alignment.count}

                    or

     SP.TAPE {printer.number} N
```

where "printer.number" is the device number of a system printer, and must be an integer between 0 and 3; and "alignment.count" is an integer greater than 0, which specifies the number of lines to be printed as alignment lines at the beginning of the report. The second format of the SP.TAPE command is used to specify that no alignment lines are to be printed.

If the "alignment.count" parameter is used, the SP.TAPE command will print "alignment.count" lines from the beginning of the tape, and will ask you to verify that the form is aligned properly in the printer. This alignment process will continue until you have verified correct alignment, and the printing of the report will then begin.

This command requires the user to insure that the specified "printer.number" (system printer 0, 1, 2 or 3) is available for assignment to your task, and that tape unit 0 is available for assignment to your task. Note that, normally, all system printers are assigned to the system SPOOL task. If the printer you wish to use is functioning as a system SPOOL printer, you will have to have your System Administrator terminate that SPOOL task before you will be able to use the SP.TAPE command with that printer.

```
     SP.TAPE {printer.number} {alignment.count}

                        or

     SP.TAPE {printer.number} N
```

FIGURE 1. Format of the SP.TAPE command.

```
:SP.TAPE 1 15
```

This command will assign system printer 1, and tape drive Ø. Tape unit Ø must have a tape mounted on it which was created by using the SETPTR command, Mode 4. The SP.TAPE processor will read this tape, and will repeatedly print 15 alignment lines, until the user verifies (in response to a prompt from SP.TAPE) that the form is properly aligned. SP.TAPE will then print the entire contents of the tape.

FIGURE 2. Example of the SP.TAPE command.

5.91 SPOOL Command

The SPOOL command allows users to:

> submit PRIME INFORMATION file records to the PRIMOS spool queue for printing;
>
> interrogate the status of the spool queue; and
>
> cancel any print files submitted by that user and still in the spool queue.

This command uses a SELECT list if one is active.

The SPOOL verb allows the user to submit records to the PRIMOS spooler for printing; list the spool queue contents; or cancel any print file submitted by that user to the PRIMOS spooler that is still in the spool queue. The three most common forms of the SPOOL command are:

```
SPOOL {file.name} {record.id} {option} ...

SPOOL -LIST

SPOOL -CANCEL nnn ...
```

where "file.name" is the name of a PRIME INFORMATION file, "record.id" is the identifier of the record to be spooled, and "nnn" is the number assigned to a print file at the time it is spooled (the number displayed by the -LIST option). The "option" parameter may be any of the options described in the PRIMOS COMMANDS REFERENCE GUIDE, FDR31Ø8, in the section on the SPOOL command.

The SPOOL command works with SELECT lists. See the section titled "SELECT AND SSELECT VERBS" in THE INFORM REFERENCE GUIDE, IDR39Ø5 for a description of SELECT lists, and the sections titled "GET.LIST Command", "SAVE.LIST Command", and "DELETE.LIST Command" in this manual.

```
SPOOL {file.name} {record.id} {option}...

SPOOL -LIST

SPOOL -CANCEL nnn ...
```

FIGURE 1. Formats of the SPOOL command.

```
:SPOOL BP MERGE
MERGE
[SPOOL rev 17.2]
Your spool file,  PRT007, is     1 record  long.

:SPOOL -LIST
[SPOOL rev 17.2]

 user  prt  time  name             size  opts/#   form  defer  at:
------ --- -----  ---------------  -----  ------  ------ -----  ------
WENDY  001 16:59  ENTRY                3
DAVID  002 17:02  REPORT1              1
WAYNE  003 17:03  MERGE                1
PEGGY  004 17:04  PAYROLL              4
ERIN   005 17:05  ORDERS               2
ED     006 17:11  PARTS                3
WAYNE  007 17:42  MERGE                1

:SPOOL -CANCEL 3
[SPOOL rev 17.2]
PRT003 has been cancelled.
```

FIGURE 2. Examples of the SPOOL command.

5.92 SSELECT (INFORM System Processor)

SSELECT is a portion of the high-level, natural language QUERY and REPORT GENERATOR portion of the PRIME INFORMATION system. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. SSELECT is one of these INFORM processors.

5.93 STAT Command

The STAT command displays information about users and resources on the system. The STAT command is an abbreviation of the STATUS command.

The STAT command is used to obtain information about the status of the entire system, such as the number of users on the system, or a listing of the disks currently mounted on the system. The STAT command is an abbreviation of the STATUS command, which is described in detail in the section of this manual titled "STATUS Command".

5.94 STATUS Command

The STATUS command displays information about users and resources on the system.

The STATUS command is used to obtain information about the status of the entire system, such as the number of users on the system, or a listing of the disks currently mounted on the system. The format of the STATUS command is:

```
     STATUS {option}
```

The options are:

USERS	displays information about the current users on the system
ME	displays information about all users on the system related to you (PHANTOM tasks)
DISKS	displays information about active disks
NETWORK	displays information about the systems in the network if PRIMENET is running
UNITS	displays a list of open disk file units
ALL	displays all the information displayed by the other options

For more details on the "NETWORK", "UNITS", and "ALL" options refer to the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108 and to the THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109.

An example of the "USERS" option is shown in Figure 2. "USER" is the login account.name of the user; "NO" is the user number; "LIN" is the port (AMLC) line number of the user terminal; "PDEV" represents the physical devices currently in use. User numbers are printed in decimal and port line numbers are printed in octal. "PDEV" includes disks that the user is using and any assigned devices. All assigned devices have the same name as those specified in the ASSIGN command with the exception of assigned disks and port lines. Assigned disks are indicated by printing the letters "DK" followed by the disk number. Assigned port lines are indicated by printing of the letters "AL" followed by the line number. Phantom users are identified by a port line number of "77". The priority level is enclosed in parentheses if it is other than the default level of 1. (See the section titled "CHAP Command" in this manual.)

If no option is specified, then the same information as the "USERS" option is displayed, except only users with the same login account.name as the user entering the STATUS command are listed.

Figure 3 is an example of the "DISKS" option. "DISK" is the disk name (packname or volume-name). "LDEV" is the logical disk number and "PDEV" is the physical disk number. "SYSN" is the name of the system on which the device is physically connected (networks only).

```
     STATUS {option}

The options are:

     USERS
     ME
     DISKS
     NETWORK
     UNITS
     ALL
```

FIGURE 1. Format of the STATUS command.

```
:STATUS USERS

USER                     NO LIN PDEVS
SYSTEM                    1  76   460 SMLC000
WAYNE                     4   2   462 MT0
DON                       9   7   462
JAN                      12  12   462 10462
SYSTEM                   13  77   460 PR0     (0)
WAYNE                    14  77   460 10462
```

FIGURE 2. Example of the STATUS command with the USERS option.

```
:STATUS DISKS

DISK    LDEV  PDEV  SYSN
MD17.3   0     460
INFO1    1     462
INFO2    2   10462
INFO3    3   20462
BMDEMO   4   71063
MD17.4   5   30462
RELSE    6   10460
```

FIGURE 3. Example of the STATUS command with the DISKS option.

5.95 SUM (INFORM System Processor)

SUM is a portion of the high-level, natural language QUERY and REPORT GENERATOR portion of the PRIME INFORMATION system. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. SUM is one of these INFORM processors.

5.96 T.DUMP (INFORM System Processor)

T.DUMP is a portion of the high-level, natural language QUERY and REPORT GENERATOR portion of the PRIME INFORMATION system. It is documented completely in THE INFORM REFERENCE GUIDE, IDR3905. This command uses a SELECT list if one is active.

Processors which are invoked by PERFORM that belong to the INFORM group are not documented in this manual. Instead, they are documented in THE INFORM REFERENCE GUIDE, IDR3905. T.DUMP is one of these INFORM processors.

5.97 TERM Command

The TERM command allows the user to set or display some of the terminal and/or line printer characteristics.

The general form of the TERM command is:

```
     TERM {A,B,C,na,na,na,G,H,na}
```

Two adjacent commas specify that the currently defined parameter is to remain unchanged. The current characteristics can be displayed by using the TERM command without a parameter list.

The parameters are (defaults are in parentheses):

A	number of characters per terminal line (80).
B	number of print lines per page on the terminal (24).
C	number of blank lines at top of page on a terminal (0). (sum of "B" and "C" equals page length)
G	number of characters per line printer line (132).
H	number of lines per line printer page (60).
na	not used, values are ignored.

The TERM command parameters "G" and "H" only affect logical print unit 0. To set other characteristics of print unit 0, see the SETPTR command in this manual. (Note that the "G" and "H" parameters of the TERM command are also affected by the SETPTR command, by the "l.len" and "p.len" parameters, respectively.)

To set other characteristics of the display terminal, see the DELAY, HUSH and PTERM commands in this manual, and the AMLC command in the THE SYSTEM ADMINISTRATOR'S GUIDE, PDR3109.

```
TERM {A,B,C,na,na,na,G,H,na}
```

The parameters are (defaults are in parentheses):

A	number of characters per terminal line (80).
B	number of print lines per page on the terminal (24).
C	number of blank lines at top of page on a terminal (0). (sum of "B" and "C" equals page length)
G	number of characters per line printer line (132).
H	number of lines per line printer page (60).
na	not used, values are ignored.

FIGURE 1. Format of the TERM command.

```
:TERM
TERM parameters are all numeric:
TERM A,B,C,D,E,F,G,H,I
   For the terminal
 A=number of characters in a line (80).
 B=number of lines per page (24).
 C=number of blank lines at top of page (0).
   For the line printer
 G=number of characters in a line (132).
 H=number of lines per page (60).

:TERM 96,55
```

FIGURE 2. Examples of the TERM command.

5.98 TIME Command

The TIME command displays the current system date and time at the user terminal.

The current date and time are printed at the user terminal with the TIME command. The format of this command is:

```
     TIME
```

For a display of the system date in a different format, see the DATE command elsewhere in this manual. For a display of time which includes total connect time for the task, CPU time used, and I/O time used, see the PTIME command elsewhere in this manual.

```
    TIME
```

FIGURE 1. Format of the TIME command.

```
:TIME

12:57:27 23 MAY 1979
```

FIGURE 2. Example of the TIME command.

5.99 UNASSIGN Command

The UNASSIGN command releases the specified peripheral allowing other users to assign it for their use.

The UNASSIGN command may be entered at the user terminal (to which a device is currently ASSIGNed) or at the supervisor terminal. The effect of this command is the opposite of ASSIGN. The UNASSIGN command, entered at the system terminal, unconditionally unassigns the peripheral assigned to any user. Entered from a user terminal, UNASSIGN only unassigns the device if it was previously assigned to the user.

The format of the command is:

```
UNASSIGN device
```

where "device" is a previously assigned peripheral. See the ASSIGN command in this manual and the ASSIGN and UNASSIGN commands in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108.

When a user logs out, all ASSIGNed peripherals are UNASSIGNed automatically by PERFORM and PRIMOS. The UNASSIGN command is useful for relinquishing control of a physical device to other users on the system, but without logging off.

```
    UNASSIGN device
```

FIGURE 1. Format of the UNASSIGN command.

```
:ASSIGN MTØ
Device MTØ  Assigned.
:UNASSIGN MT1
Device MT1  Not Assigned.
:UNASSIGN MTØ
```

FIGURE 2. Examples of the UNASSIGN command.

5.100 USERS Command

The USERS command prints the number of users currently logged into the system.

The number of users currently logged into the system is displayed by the USERS command. The format of this command is:

```
     USERS
```

The result of this command will be a single line display of the number of users currently logged into the system. Note that the "STATUS USERS" command (described in the section titled "STATUS Command" in this manual) will always show user number 1, on line number 76, in addition to the other users on the system. User number 1 is the system console task, and is not included in the count displayed by the USERS command. Note also that the USERS command does not distinguish between a user who is actively using the system, and one who has been gone for several hours but is still logged in.

The STATUS command will display more information about the users. See the section titled "STATUS Command" in this manual. The PRIME INFORMATION system also includes a stored sentence, LISTU, which displays information about all users currently sharing the system.

```
    USERS
```

FIGURE 1. Format of the USERS command.

```
:USERS

USERS = 14
```

FIGURE 2. Example of USERS command.

5.101 VCATALOG Command

The VCATALOG command allows a user to compare the object code in a program source and object file to the CATALOGed object code.

When a program is CATALOGed on a PRIME INFORMATION system, the object code is taken from the file where the source and object code are stored, and entered into the catalog space. If the original object code is retained after the CATALOG operation, the catalog version of that program may be verified at any time by the VCATALOG command. The format of the command is:

```
     VCATALOG
```

If the object code in the file has not been changed since the program was CATALOGed, the VCATALOG processor will display the message "PROGRAM VERIFIES". If it does not display this message, then the version of the program in the catalog is not the version that was CATALOGed from the object code in the file.

Note that creating a new copy of the object code in the file by recompiling the original source code is sufficient to cause the VCATALOG operation to indicate that the program does not verify. The verification process encompasses not just the executable object code (which would be identical in this example), but also the date and time that it was compiled (which would not be identical in this example).

VCATALOG

FIGURE 1. Format of the VCATALOG command.

```
:CATALOG
Catalog name =*SOUNDEX
File name =PROGRAMS
Program name =SOUNDEX
*SOUNDEX cataloged

:VCATALOG
Catalog name =*SOUNDEX
File name =PROGRAMS
Program name =SOUNDEX
Program verifies

:BASIC PROGRAMS SOUNDEX
SOUNDEX
**********

    0 Error(s).    100 Lines.
(1) Program compiled

:VCATALOG
Catalog name =*SOUNDEX
File name =PROGRAMS
Program name =SOUNDEX
SEGNO 002006, WORD 047035, FILE = 147307, AND CATALOG SPACE = 000000
Program does not verify, code=3.
```

FIGURE 2. Examples of the VCATALOG command.

5.102 VVOC Command

The VVOC command is used to verify the contents of your VOC file, by comparing your VOC file to the system NEWACC file.

The VVOC command is used to verify the contents of your VOC file, by comparing your VOC file to the system NEWACC file. The VOC file of each account is created at the time the account is created, by copying the current contents of the NEWACC file into the new VOC file. (See the description of the INFO command and the LOGTO command to see how a PRIME INFORMATION account is created.) After it has been created, a VOC file will have many new items added to it which are particular to that account, and therefore will not appear in the NEWACC file.

When the Release level of the PRIME INFORMATION software is changed, you will be informed that your account is no longer up-to-date when you first enter it (via either the PRIMOS-level INFO or LOGIN commands, or via the PRIME INFORMATION-level LOGTO command). If your account's VOC file is out of date, and if you confirm that you wish it to be updated, it will be updated to the latest release level by again copying items from the NEWACC file into your VOC file. However, there is no mechanism except this update process that will confirm that your VOC file matches the current contents of the NEWACC file; additionally, even the update process will not eliminate items from your VOC file that have been superceded by new commands. The VVOC command is provided to help you assure that your VOC file is up-to-date. The format of this command is:

```
     VVOC {NO.PAGE} {LPTR}
```

This command will select all of the items in your account's VOC file, and all of the items in the NEWACC file, and compare them one by one. If any items are encountered in your account's VOC file which do not exist in the NEWACC file, they will be listed on your terminal (or on the printer if you specified the optional "LPTR" keyword). Each such item will also be put into a SELECT list (in the &SAVEDLISTS& file), with the name "VOC.ONLY000"; this SELECT list can be activated by the command "GET.LIST VOC.ONLY". Note that there will surely be some of these, including items such as local FILE definition records, your own stored SENTENCE and PARAGRAPH records, MENU records, etc.

If any items are encountered in the NEWACC file which do not exist in your account's VOC file, they will be listed and put into a SELECT list named "NEWACC.ONLY000"; this SELECT list can be activated by the command "GET.LIST NEWACC.ONLY". You should examine this list very carefully, because in most installations there should be none of these. However, your System Administrator may have deliberately removed some items from most account VOC files, so you should verify with your System Administrator that any items listed actually denote an error.

Finally, if any items are encountered that exist in both your account's VOC file and the NEWACC file, but which are different in these two files, they will be listed, and put into a SELECT list named "VOC.DIFF000"; this SELECT list can be activated by the command "GET.LIST VOC.DIFF000". Any items in this list are particularly likely to be indicators of error, but (as above) you should verify this with your System Administrator; the differences may be intentional, depending on the specific needs of your organization.

```
     VVOC {NO.PAGE} {LPTR}
```

FIGURE 1. Format of the VVOC command.

```
:VVOC
SSELECT VOC
322 records selected.
SSELECT NEWACC
319 records selected.
The following items appear in your VOC file, but not in NEWACC.
 12:03:29  11-06-80  Page 1

&ED&
&PH&
&SAVEDLISTS&


The following items are different in your VOC file than in NEWACC.
12:03:29  11-06-80  Page 1

&MAP&

There are 3 items in your VOC file that are not in the NEWACC file.
These items were saved as SELECT list VOC.ONLY.

1 items in your VOC file do not match those in the NEWACC file.
These items were saved as SELECT list VOC.DIFF.
```

FIGURE 2. Example of the VVOC command.

5.103 WHO Command

The WHO command displays the port number and account name and, where the account name and login name are different, displays the login name.

The WHO command displays the user's port number, followed by the account name where the user is currently active. If the account name is different from the login name (as it would be after executing the LOGTO command within the PRIME INFORMATION environment, or the ATTACH command within the PRIMOS environment), the WHO command additionally displays "FROM login.name".

The format of the command is:

```
    WHO
```

The port number is a constant for any given terminal on the system: i.e., the terminal is physically connected to a port on the system. However, the account name may be changed during the course of processing by any of several events, particularly the PRIMOS ATTACH, or the PRIME INFORMATION LOGTO.

At the time of login, PRIMOS captures and maintains (for the life of the task) the account name that was used with the LOGIN command. When PRIME INFORMATION is entered from the PRIMOS environment (an event that, on most systems, will be done automatically during LOGIN), PERFORM retrieves from PRIMOS the account name from LOGIN, and the name of the account where the user is currently attached. (Normally, these are the same, but they need not be; for example, the PRIMOS-level ATTACH command can modify where the user is attached after login, without PRIME INFORMATION being aware of this at the time of the ATTACH.)

During the course of processing on the system, a user may use the LOGTO command to move to a different account. If so, PERFORM will automatically set the account name returned by the WHO command to the account name used with the LOGTO command, and will append to the display line the message "FROM login.name", where "login.name" is the original account name used when the user logged in. Thus, the user is always identifiable by the original LOGIN account name, regardless of which account is currently being used by that user.

```
     WHO
```

FIGURE 1. Format of the WHO command.

```
:WHO

2 MASTER
```

The user who issued the WHO command is on terminal port number 2, and is using the account named MASTER. Since there is no additional data displayed, this user also logged into the account named MASTER.

FIGURE 2. Example of the WHO command.

```
:WHO

7 ACCOUNTING   FROM PAYROLL
```

The user who issued the WHO command is on terminal port number 7, and is using the account named ACCOUNTING. This user originally logged into the account named PAYROLL.

FIGURE 3. Example of the WHO command after the LOGTO command.

---

6 PARAGRAPHS AND SYSTEM CONTROL

6.1 Prompting Within a Stored PERFORM Command

In some stored sentences, paragraphs and phrases, there are items of information that must be filled in at the time the sentence, paragraph or phrase is used. This is accomplished through the use of inline prompting.

Inline prompting is a mechanism by which parts of a sentence may be left unspecified until the sentence is actually executed. At that time, the prompt defined within the stored sentence will be displayed on the terminal, and the data entered in response to the prompt will be used as if it had been included in the stored sentence.

Inline prompting is accomplished by including in the command, at the point where it will be used, a prompt string that is of the form:

```
     << {control,}...  prompt.text {,check} >>
```

When PERFORM encounters the prompt, the "control" options, if present, are performed and then the "prompt.text" is printed to request input from the user. The input from the terminal is then checked against the "check" option if present. If what was entered from the keyboard does not match the specified check, the prompt for input is re-issued.

If the same "prompt.text" occurs more than once in a stored paragraph, the input to the first "prompt.text" is stored by PERFORM and automatically substituted each time the "prompt.text" is encountered without reprompting, unless the "A" control option (for "Always prompt") is included in the prompt.

The "Cn" control option allows the user to type the inputs for the prompts on the same line with the sentence or paragraph name. (Note that "prompt.text" is required with the "Cn" option.)

The "In" option is the same as the "Cn" option, except that PERFORM prompts if the value is null.

If "QUIT" is entered in response to a prompt, the sentence or paragraph will abort back to PERFORM's colon prompt.

There can be prompts within prompts. For example:

```
     <<F(<<ENTER NAME OF FILE>>,<<ENTER RECORD ID>>),prompt.text>>
```

```
<<{control,}... prompt.text {,check}>>
```

The "control" options are:

@(CLR)	Clear the terminal screen.
@(BELL)	Ring the terminal bell.
@(TOF)	Issue a top-of-form.
@(column,row)	Prompt at screen position "column", "row".
R	Repetitively prompt until a null is entered.
R(string)	Repetitively prompt until a null is entered and insert "string" between each entry.
A	Always prompt, never automatically fill.
Cn	Take word "n" from the command line entered to execute the sentence or paragraph.
In	Take word "n" from the command line (as in the "Cn" option), but prompt if word "n" was not entered on the command line.
F(fn,rn{,fm,vm,sm})	Get prompt from record "rn" in file "fn" and optionally extract from field "fm", value "vm", and sub-value "sm". When this option is used the "prompt.text" is optional.

The "check" options can be any pattern match or any valid conversion. The conversion must be enclosed in parentheses.

FIGURE 1. Format of the Inline Prompt.

<<ENTER QUANTITY>>	Simplest form.
<<ENTER PART NUMBER,6N>>	Pattern check for 6 numerics.
<<DATE OF ORDER,(D)>>	Pattern check for any date.
<<@(CLR),@(40,11),QUANITY,0N>>	Clear screen, prompt at screen column 40 and row 11, and verify that input is numeric.
"<<R(" "),PART NUMBERS,5N-2A>>"	Repetitively prompt for numbers of the form 5 numeric, dash, 2 alpha, enclosing each entry in quotes.

FIGURE 2. Examples of Inline Prompts.

6.2 DATA Statements in Paragraphs

At times it is advantageous to be able to pre-store responses to expected requests for data in a paragraph. These pre-stored responses may be used in place of terminal input by INFO/BASIC programs and PERFORM verbs. The DATA command is provided for this purpose.

When a paragraph is used to invoke either an INFO/BASIC program or a PERFORM verb, it is often desirable to provide within the paragraph responses to requests for data input that the program or verb will make. The DATA command allows the user to store such responses in the paragraph. The format of the command is as follows:

```
DATA response.data
```

This causes "response.data" to be stacked as input to a subsequent program or verb. When the program or verb requires data input that would normally be provided by a user at a terminal, the first DATA statement's response.data is provided instead. The next request for data input is satisfied by the next DATA statement's response.data, and so on, until there are no more DATA statements in the paragraph. Requests for data input after all DATA statements have been used will operate as usual (i.e., the data will be entered by the user at the terminal).

Note that the INFO/BASIC program name or the PERFORM verb name will appear on one line of the paragraph, and the DATA statements to be used by that program or verb will appear on <u>subsequent</u> line(s).

Inline prompting can also be used with the DATA statement, illustrated by the example in Figure 2. (See also the section titled "Prompting Within a Stored PERFORM Command" elsewhere in this manual.)

```
     DATA response.data
```

where "response.data" is the data to pass to the INPUT statement.

FIGURE 1. Format of the DATA statement in a paragraph.

```
      id:  BACKORDER

     001:  PA
     002:  RUN MASTER ORDER.MAINTENANCE
     003:  DATA 10.15
     004:  DATA <<ENTER QUANITY,0N>>
     005:  DATA YES
     006:  RUN MASTER ORDER.LIST
     007:  DATA NO
```

FIGURE 2. Examples of the DATA statement in a paragraph.

6.3 IF . . . THEN GO Paragraph Control Structure

The IF . . . THEN GO paragraph control structure is used to modify the sequence of execution of the sentences within a PRIME INFORMATION paragraph.

The IF . . . THEN GO paragraph control structure is used within PRIME INFORMATION paragraphs to alter the sequence of execution of the sentences in the paragraph, depending on the outcome of a test. The IF . . . THEN GO structure is always necessary within a LOOP . . . REPEAT control structure, since there is otherwise no way to terminate the loop; however, the IF . . . THEN GO structure may be used at other points in a paragraph as well. The format of the IF . . . THEN GO control structure is:

```
     IF condition.expression THEN GO statement.label
```

where "condition.expression" is the test to be performed and "statement.label" is the label of a sentence which comes after the IF . . . THEN GO structure and is within the same paragraph.

At the time of this writing, the "condition.expression" must adhere to a set of fairly rigid rules; this rigidity may be relaxed in the future. The "condition.expression" may be illustrated by the following model:

```
     value.1 relational.operator value.2
```

where "value.1" and "value.2" are the two values to be compared, and "relational.operator" defines the type of comparison to be made: for example, "A = B", "X <= Y", or "NAME MATCHES JON...". Within the IF . . . THEN GO paragraph control structure, "value.1" can only be the result obtained by a PROMPT operation (defined in the section on PROMPTING in this manual), and "value.2" can only be a constant (a literal string enclosed in quotation marks). Since the PROMPT will be executed repeatedly, you must remember to use the "ALWAYS PROMPT" option; if you forget it, the prompt will be issued only the first time it is encountered, and there will be no opportunity to exit the loop.

Note also that "statement.label" must be a label within the same paragraph, and that the labelled statement must come after the IF . . . THEN GO control structure within that paragraph: "backward" destinations are not permitted.

```
LOOP
{loop.exit}
sentence.1
sentence.2
sentence.3
sentence.n
{loop.exit}
REPEAT
```

FIGURE 1. Format of the IF . . . THEN GO paragraph control structure.

```
PARAGRAPH
LOOP
IF <<A,Enter a new file name>> = '' THEN GO FINISHED
LIST DICT <<Enter a new file name>> LPTR
GROUP.STAT.DETAIL DICT <<Enter a new file name>> LPTR
LIST <<Enter a new file name>> LPTR
GROUP.STAT.DETAIL <<Enter a new file name>> LPTR
REPEAT
FINISHED: DISPLAY "Finished file listings."
```

FIGURE 2. Example of the IF . . . THEN GO paragraph control structure.

6.4 Labels in PERFORM Paragraphs

Each sentence in a PERFORM paragraph may have a label to uniquely identify it. A labelled statement may be used as the branch destination of the IF . . . THEN GO control structure.

Each individual sentence of a PERFORM paragraph may optionally have a label which uniquely identifies it within the paragraph. Labelled sentences in a PERFORM paragraph may be used as the branch destination of the IF . . . THEN GO paragraph control structure, which is defined elsewhere in this manual. The format of a PERFORM paragraph statement label is:

```
     label.name:  sentence.text
```

where "label.name" may be solely numeric (digits 0 through 9), or may be alphanumeric (begin with a letter, optionally followed by any digits, letters, dollar signs, or periods), and must end with a colon (:). There must be at least one space between the colon which marks the end of the label, and the beginning of the sentence.

```
     label.name: sentence.text
```

FIGURE 1. Format of Labels in PERFORM Paragraphs.

```
001: PA
002: IF <<A,Enter file name>> = '' THEN GO NEXT.TASK
003: IF <<A,DICT only (Y or N),1A>> = 'Y' THEN GO DICT.ONLY
004: SORT <<Enter file name>> ID.ONLY LPTR
005: DICT.ONLY: LIST DICT <<Enter file name>> LPTR
006: NEXT.TASK: IF <<A,End of Quarter (Y or N)>> = 'N' THEN GO FINISHED
007: EOQ.POSTING
008: FINISHED: DISPLAY Finished.
```

FIGURE 2. Examples of Labels in PERFORM Paragraphs.

6.5 LOOP . . . REPEAT Paragraph Control Structure

The LOOP . . . REPEAT paragraph control structure is used to define a range of sentences which is to be executed repeatedly within a PRIME INFORMATION paragraph.

The LOOP . . . REPEAT paragraph control structure is used within PRIME INFORMATION paragraphs to define a range of sentences which is to be executed repeatedly. The keyword LOOP is used to introduce the beginning of the group of sentences, and the keyword REPEAT is used to mark the end of the group. Together, the LOOP . . . REPEAT keywords define the scope of the repetitive loop. The format of the LOOP . . . REPEAT control structure is:

```
LOOP
{loop.exit}
sentence.1
sentence.2
sentence.3
sentence.n
{loop.exit}
REPEAT
```

where "sentence.1" through "sentence.n" is the group of sentences to be executed repeatedly, and "loop.exit" is the means by which the loop will be terminated. The "loop.exit" statement may appear anywhere within the scope of the loop, and there may be more than one "loop.exit" statement; however, normally the "loop.exit" statement will appear either as the first statement after the LOOP keyword, or as the last statement before the REPEAT keyword. The "loop.exit" statement is represented by the "IF . . . THEN GO" paragraph control structure, which is described elsewhere in this section.

Note that failure to include a mechanism for exit from the LOOP . . . REPEAT paragraph control structure may cause a task to loop indefinitely.

```
LOOP
{loop.exit}
sentence.1
sentence.2
sentence.3
sentence.n
{loop.exit}
REPEAT
```

FIGURE 1. Format of the LOOP . . . REPEAT paragraph control structure.

```
PARAGRAPH
LOOP
IF <<A,Want to continue loop,1A>> = "N" THEN GO SKIP
DISPLAY "This is statement 1 of LOOP/REPEAT sequence"
DISPLAY "This is statement 2 of LOOP/REPEAT sequence"
DISPLAY "This is statement 3 of LOOP/REPEAT sequence"
DISPLAY "This is statement 4 of LOOP/REPEAT sequence"
DISPLAY "This is statement 5 of LOOP/REPEAT sequence"
DISPLAY "This is statement 6 of LOOP/REPEAT sequence"
REPEAT
SKIP: DISPLAY "This is destination of GO statement"
DISPLAY "All finished"
```

FIGURE 2. Example of LOOP . . . REPEAT paragraph control structure.

6.6 LOGIN Procedure in VOC File

Upon initial entry into a PRIME INFORMATION account from another account or from the PRIMOS environment, PERFORM looks for an item in the VOC file with the name "LOGIN". If this item is found, and if it is a paragraph, sentence, menu, or verb, PERFORM will execute it before issuing its ":" prompt.

When a PRIME INFORMATION account is initially entered, whether from the PRIMOS environment (by the PRIMOS-level command LOGIN or INFO) or from the PRIME INFORMATION environment (via the PERFORM command LOGTO), PERFORM searches the account's VOC file for an item named LOGIN. This item, if present, must be a sentence, a paragraph, a menu, or a verb. The LOGIN item is defined by the user, and may be any valid sequence of operations that are normally available to PRIME INFORMATION users.

The LOGIN procedure is most often used for any of the following purposes:

to invoke a menu of operations;

to invoke a security checking and/or system accounting program;

to set system parameters (such as SETPTR, DELAY, etc.);

or to automatically invoke a processing program.

```
 id: LOGIN

001: PA
002: SETPTR 0,120,66,5,3,1,FORM LEDGER,BRIEF
003: IAM ACCTNG
004: GEN.LEDGER.MENU
005: OFF
```

This LOGIN procedure sets the printer characteristics for the form named LEDGER; establishes the correct account.name (ACCTNG); invokes the menu for General Ledger posting; and automatically logs the user off the system when the menu operations are completed.

FIGURE 1. Example of LOGIN procedure.

```
 id: LOGIN

001: PA
002: IAM DEMO
003: SETPTR 0,80,66,3,3,1,FORM WHITE,BRIEF
004: DISPLAY ***************************************************
005: DISPLAY **  Welcome to PRIME INFORMATION Release _
006:  <<F(VOC,RELLEVEL,2),RELLEVEL>> **
006: DISPLAY ***************************************************
007: RUN BP SYSTEM.ACCOUNTING
008: DISPLAY Type your commands now
```

This LOGIN procedure sets the account.name for the DEMO account; sets the printer characteristics for 8 1/2 X 11 inch white paper; displays a "welcome" message (which extracts the release level from the VOC file via an "invisible" prompt sequence); runs the accounting program for system usage accounting; and allows the user to control the remainder of activity in the account.

To understand the "invisible" prompt sequence of this example, see the section titled "Prompting Within a Stored PERFORM Command" in this manual.

FIGURE 2. Example of LOGIN procedure.

---

7 PERFORM SENTENCE STACK

7.1 PERFORM Sentence Stack Introduction

PERFORM internally stores the last 99 sentences a user has typed at PERFORM's ":" prompt. Any sentence in this stack can be re-executed, changed, or stored as a sentence or paragraph in the VOC file. Optionally, this stack can be retained by PERFORM even when you leave the PRIME INFORMATION environment (by QUIT or LOGOUT).

Each time you type a sentence at PERFORM's ":" prompt, PERFORM saves the sentence on a stack which is managed automatically by PERFORM. Up to 99 sentences are retained on the stack. As each new sentence is added to the sentence stack, each previous sentence is "pushed up", with the sentence which was number 99 being "pushed off" the sentence stack, and the new sentence becoming sentence number 1.

The sentences in your sentence stack can be recalled at any time for your review, with the sentence stack LIST command. With this command, you can easily review the sequence of operations which you have recently performed. Additionally, you can modify any sentence which is on your sentence stack, with the CHANGE and APPEND commands. You might wish to do this, for example, to append "LPTR" to a sentence which you had executed earlier, or to change "LIST DICT CUSTOMERS" to "LIST CUSTOMERS". After modifying any sentence, it is then possible to re-execute that sentence with the sentence stack EXECUTE command.

It is also possible to remove sentences from your sentence stack, with the DELETE command, or to add new sentences to your sentence stack (without having PERFORM execute or even evaluate them for correctness) with the INSERT command. Either or both of these commands might be useful in preparation for using yet another sentence stack command, SAVE, which allows you to save portions of your sentence stack as a VOC file PARAGRAPH (saving multiple sentences from your sentence stack), or as a VOC file SENTENCE (saving a single sentence from your sentence stack).

Other operations which you can perform with the sentence stack commands are to LIST the contents of a VOC file SENTENCE or PARAGRAPH, or to RECALL the contents of a VOC file SENTENCE or PARAGRAPH into your sentence stack. The LIST operation simply displays the SENTENCE or PARAGRAPH, but does not affect your sentence stack, while the RECALL command puts the SENTENCE or PARAGRAPH into your sentence stack (where it may be modified with the APPEND or CHANGE commands, and then EXECUTED).

As you are typing a sentence at the PERFORM ":" prompt, you may notice that you have made an error earlier in the sentence. Although you can always backspace to the error, correct it, and re-type the remainder of the sentence, this can be a time-consuming operation, particularly if the error is at the beginning of a lengthy sentence that is nearly finished when you discover your mistake. In this situation, you can merely terminate the sentence with a question mark (?). When PERFORM encounters the question mark, it simply places the sentence on your sentence stack, but does not execute it and does not even evaluate it for correctness. This is much like the INSERT sentence stack command, except that it was not initially planned by you! Now you can use the sentence stack CHANGE command to correct your error, and the EXECUTE command to direct PERFORM to execute the (now correct) sentence.

Since there is such a rich variety of sentence stack operations available to you, you may forget what they are, or how to use them. Still another sentence stack command is provided to help you remember the other commands: the ".?" command. At PERFORM's ":" prompt, type a period and a question mark (.?) and PERFORM will display a list of all the sentence stack commands.

The following pages contain a detailed description of each of the sentence stack commands.

.A# any	Append "any" to end of sentence "#"
.C#/s1/s2/	Change "s1" to "s2" in sentence "#"
.D#	Delete sentence number "#" from stack
.D name	Delete sentence or paragraph "name" from VOC
.I# any	Insert "any" in stack before sentence "#"
.L#	List "#" lines of stack (default is 20)
.L name	List paragraph or sentence "name" from VOC
.R#	Recalls sentence number "#" to number 1
.R name	Loads "name" from VOC into stack
.S name s# e#	Save lines s# thru e# in VOC as "name"
.X#	Executes sentence number "#"
.?	Display the sentence stack commands
:SENTENCE WITH MISSPELLING?	Put sentence into PERFORM sentence stack for correction, but do not execute it

FIGURE 1. A summary of the Dot commands.

7.2 .? (Display Sentence Stack Commands)

A complete list of the PERFORM sentence stack commands may be requested by typing ".?" at the PERFORM ":" prompt.

Each of the PERFORM sentence stack commands is described in detail in this section of this manual. However, it is easy to forget exactly which options you have in manipulating the PERFORM sentence stack, or the order or format of the parameters of each sentence stack command. To help you remember how to use the PERFORM sentence stack command, PERFORM provides a summary listing of each command in response to ".?" at the ":" prompt.

This request for help has no operands nor parameters. In response to the ".?", PERFORM will display a list of every sentence stack command, its operands, and the function each command will perform.

```
     .?
```

FIGURE 1. Format of the HELP sentence stack command.

```
:.?
```

.A# any	Append "any" to end of sentence "#"
.C#/s1/s2/	Change "s1" to "s2" in sentence "#"
.D#	Delete sentence number "#" from stack
.D name	Delete sentence or paragraph "name" from VOC
.I# any	Insert "any" in stack before sentence "#"
.L#	List "#" lines of stack (default is 20)
.L name	List paragraph or sentence "name" from VOC
.R#	Recalls sentence number "#" to number 1
.R name	Loads "name" from VOC into stack
.S name s# e#	Save lines s# thru e# in VOC as "name"
.X#	Executes sentence number "#"

FIGURE 2. Example of the HELP sentence stack command.

7.3 .A (Append to Sentence in PERFORM Sentence Stack)

Any sentence in the PERFORM sentence stack may have additional words appended to the end of it by using the ".A" stack command.

The ".A" sentence stack command may be used to append additional operators to any sentence in the PERFORM sentence stack. For example, after testing a complicated LIST statement and viewing its results on the terminal for correctness, it is easy to simply append " LPTR" to the sentence and re-execute it (with the ".X" command, described elsewhere in this manual) to get a printed report.

The format of the ".A" sentence stack command is:

```
.A# any.text
```

where "#" is the number of the sentence in the stack to which the text "any.text" is to be appended. If "#" is omitted, the text "any.text" is appended to sentence number 1 in the sentence stack by default.

The text "any.text" should appear exactly as it would if it had been typed as part of the sentence originally. That is, it is not necessary to enclose this text in quotation marks (unless the inclusion of it in the desired sentence would have required quotation marks). Note that this command does not automatically insert a space at the end of the original sentence before the text "any.text" is added; if a space is necessary here (as it usually is), you must explicitly type <u>two</u> spaces after the ".A" or ".A#" command, and before the text "any.text". For example:

```
.L1
1 LIST STATES WITH SETTLED > 1699
.A AND SETTLED < 1800
```

would produce a sentence (not the one desired) as follows:

```
LIST STATES WITH SETTLED > 1699AND SETTLED < 1800
```

The correct ".A" command for this example would be:

```
.A  AND SETTLED < 1800     (note the extra space after .A)
LIST STATES WITH SETTLED > 1699 AND SETTLED < 1800
```

```
.A# any.text
```

FIGURE 1. Format of the APPEND sentence stack command.

```
:.L1
1 LIST EMPLOYEES
:.A  WITH DEPENDENTS > 3
LIST EMPLOYEES WITH DEPENDENTS > 3
:.A  AND SALARY < 1000
LIST EMPLOYEES WITH DEPENDENTS > 3 AND SALARY < 1000
:.A 0
LIST EMPLOYEES WITH DEPENDENTS > 3 AND SALARY < 10000
:.A  LPTR
LIST EMPLOYEES WITH DEPENDENTS > 3 AND SALARY < 10000 LPTR
```

FIGURE 2. Examples of the APPEND sentence stack command.

7.4 .C (Change Sentence in PERFORM Sentence Stack)

Any sentence in the PERFORM sentence stack may be changed by using the ".C" stack command.

The ".C" sentence stack command may be used to modify any sentence in the PERFORM sentence stack. For example, after testing a complicated LIST statement and viewing its results on the terminal for correctness, it is easy to change it to reflect a new sorting order specification, or to change the data elements which will be displayed.

The format of the ".C" sentence stack command is:

```
.C#/s1/s2/{G}
```

where "#" is the number of the sentence in the stack which is to be modified. If "#" is omitted, the modification is done to sentence number 1 in the sentence stack by default.

The remainder of the ".C" command is the text string which is to be changed ("s1" in the format example), and the text string which is to replace "s1" ("s2" in the format example), separated by common delimiters ("/" in the format example). These common delimiters may actually be chosen from a number of special characters, listed below:

```
!  "  #  $  %  &  (  )  *  +  ,  -  .

/  :  =  @  [  ]  \  _  `  '  |  {  }
```

The optional character "G" at the end of the ".C" sentence stack command specifies that the change is to be done globally on the affected sentence. For example, the sentence:

```
COPY FROM DICT MASTER.FILE TO DICT BACKUP.MASTER ALL OVERWRITING
```

could be changed as follows:

```
:.C/ DICT//G
COPY FROM MASTER.FILE TO BACKUP.MASTER ALL OVERWRITING
```

```
.C#/s1/s2/{G}
```

FIGURE 1. Format of the CHANGE sentence stack command.

```
:.L1
1 LIST EMPLOYEES BY DATE.EMPLOYED BY SALARY
:.C/DATE.EMPLOYED/JOB.LEVEL/
LIST EMPLOYEES BY JOB.LEVEL BY SALARY
:.C'BY'BY.DSND'
LIST EMPLOYEES BY.DSND JOB.LEVEL BY SALARY
:.C+BY SALARY+WITH SALARY < 15000+
LIST EMPLOYEES BY.DSND JOB.LEVEL WITH SALARY < 15000
:.C!15!21!
LIST EMPLOYEES BY.DSND JOB.LEVEL WITH SALARY < 21000
```

FIGURE 2. Examples of the CHANGE sentence stack command.

7.5 .D (Delete Sentence from PERFORM Sentence Stack)

Any sentence in the PERFORM sentence stack may be deleted from the stack by the ".D" stack command. Additionally, any sentence or paragraph in the VOC file may be deleted from the VOC file by the ".D" stack command.

The ".D" stack command has two modes, depending on how it is used. The first mode is invoked by the format:

```
    .D#
```

where "#" is the number of a sentence in the PERFORM sentence stack which is to be deleted. This form of the command is useful in preparing a section of the PERFORM sentence stack to be saved as a PARAGRAPH in the VOC file. (See the section on the ".S" command.)

The second mode of the ".D" stack command is invoked by the format:

```
    .D name
```

where "name" is the name of a SENTENCE or PARAGRAPH in the VOC file which is to be deleted from the VOC file. This form of the ".D" command is most likely to be used after first examining the contents of the sentence or paragraph to be deleted, which can be accomplished easily with the ".L" command (described elsewhere in this manual).

.D#	Delete sentence number "#" from the PERFORM sentence stack
.D name	Delete the sentence or paragraph whose name is "name" from the VOC file

FIGURE 1. Formats of the DELETE sentence stack command.

```
:.L
4 SELECT CUSTOMERS WITH LAST.ORDER BEFORE 1/15/80
3 SELECT CUSTOMERS BY ZIP.CODE
2 RUN PROGRAMS MAIL.LIST
1 LIST EMPLOYEES WITH DEPENDENTS > 3 BY SALARY NAME SALARY DEPENDENTS
:.D3
:.L
3 SELECT CUSTOMERS WITH LAST.ORDER BEFORE 1/15/80
2 RUN PROGRAMS MAIL.LIST
1 LIST EMPLOYEES WITH DEPENDENTS > 3 BY SALARY NAME SALARY DEPENDENTS
```

FIGURE 2. Example of the DELETE sentence stack command.

7.6 .I (Insert Sentence Into PERFORM Sentence Stack)

A new sentence may be inserted into the PERFORM sentence stack at any location within the stack by using the ".I" stack command.

The ".I" sentence stack command may be used to insert a new sentence into the PERFORM sentence stack. This insertion command may be used to add a new sentence into a sequence of related sentences already on the stack, since it is possible to specify where the new sentence is to be inserted.

The format of the ".I" sentence stack command is:

```
      .I# any.text
```

where "#" is the number of the sentence in the stack where the new sentence "any.text" is to be inserted. Any sentence already in the stack with a number equal to or higher than "#" will have its number increased by one, while sentences in the stack with numbers lower than "#" will retain their original number. If "#" is omitted, the sentence "any.text" is inserted as sentence number 1 in the sentence stack by default.

The "any.text" parameter should be a complete sentence that would be acceptable at the PERFORM ":" prompt. Presumably, the new sentence is being inserted into the sentence stack to prepare a block of sentences to be saved as a PARAGRAPH in the VOC file (see the description of the .S command elsewhere in this section), and therefore the sentence must be valid. PERFORM does not check the validity of the sentence before inserting it into the sentence stack.

```
.I# any.text
```

FIGURE 1. Format of the INSERT sentence stack command.

```
:.L
05 ED STATISTICS MOVING.AVERAGE
04 BASIC STATISTICS MOVING.AVERAGE
03 RUN STATISTICS MOVING.AVERAGE
02 CATALOG STATISTICS MOVING.AVERAGE
01 RUN STATISTICS BAR.GRAPH
:.I4 COPY FROM MASTER.STATS TO WORK.STATS ALL OVERWRITING
COPY FROM MASTER.STATS TO WORK.STATS ALL OVERWRITING
:.L
06 ED STATISTICS MOVING.AVERAGE
05 BASIC STATISTICS MOVING.AVERAGE
04 COPY FROM MASTER.STATS TO WORK.STATS ALL OVERWRITING
03 RUN STATISTICS MOVING.AVERAGE
02 CATALOG STATISTICS MOVING.AVERAGE
01 RUN STATISTICS BAR.GRAPH
```

FIGURE 2. Example of the INSERT sentence stack command.

7.7 .L (List PERFORM Sentence Stack)

The PERFORM sentence stack may be listed by the ".L" stack command. Additionally, any stored system SENTENCE or PARAGRAPH may be listed by the ".L" command.

The PERFORM sentence stack may be displayed at the terminal by the ".L" sentence stack command. Additionally, the ".L" stack command may be used to display at the terminal the contents of any stored system SENTENCE or PARAGRAPH in the VOC file. The format of the ".L" command used determines which of these two events will occur. The format of the ".L" sentence stack command which produces a display of PERFORM's sentence stack is:

```
.L#
```

where "#" is the number of sentences in the stack which are to be displayed. This number may be from 1 to 99 or, if omitted, will be taken as 20 (the default value, chosen because it fits well on a CRT display).

The format for the ".L" stack command which lists the stored system SENTENCE or PARAGRAPH from the VOC file is:

```
.L name
```

where "name" is the name of an item in the VOC file whose type is either "S" (for SENTENCE) or "PA" (for PARAGRAPH). When this form of the ".L" command is used, the named VOC item is displayed at the terminal, but is not put into the PERFORM sentence stack. This command is useful for quickly checking the actual contents of a sentence or paragraph before executing it. A related sentence stack command is the RECALL command (".R") described elsewhere in this section.

.L#	List "#" sentences from the PERFORM sentence stack. (Default for "#" is 20.)
.L name	Display the contents of the VOC file SENTENCE or PARAGRAPH whose id is "name".

FIGURE 1. Formats of the LIST sentence stack command.

```
:.L3
03 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
02 LIST ORGANIZATION EMPLOYEE.ID
01 LIST EMPLOYEES BY DEPARTMENT

:.L LISTS

Sentence to list all STORED SENTENCES in the VOCABULARY file
SORT VOC WITH TYPE = "S" NAME DESC F2 ID.SUP DBL.SPC_
HEADING "SENTENCES stored in your VOCABULARY file"

3 lines listed from VOC file.
```

FIGURE 2. Examples of the LIST sentence stack command.

7.8 .R (Recall Sentence/Paragraph Into PERFORM Sentence Stack)

Any sentence in the PERFORM sentence stack may be recalled into the position of sentence number one by the ".R" stack command. Additionally, any stored system SENTENCE or PARAGRAPH may be loaded from the VOC file into the PERFORM sentence stack by the ".R" command.

Any sentence in the PERFORM sentence stack may be recalled as sentence number one in the stack by the ".R" sentence stack command. Additionally, the ".R" stack command may be used to recall into the PERFORM sentence stack the contents of any stored system SENTENCE or PARAGRAPH in the VOC file. The format of the ".R" command determines which of these two events will occur.

The format of the ".R" sentence stack command which recalls a sentence from the PERFORM sentence stack into sentence number one is:

```
.R#
```

where "#" is the number of the sentence in the stack which is to be recalled. This number may be from 1 to 99 or, if omitted, will be taken as 1 (the default value: note that this would be a "no-operation" action).

The format for the ".R" stack command which recalls a SENTENCE or PARAGRAPH from the VOC file into the PERFORM sentence stack is:

```
.R name
```

where "name" is the name of an item in the VOC file whose type is either "S" (for SENTENCE) or "PA" (for PARAGRAPH). When this form of the ".R" command is used, the named VOC item is put into the PERFORM sentence stack as sentence number one (for a VOC sentence) or as sentences numbers one through "n" (where "n" is the number of sentences in the VOC file paragraph). This form of the ".R" command is useful for quickly changing the contents of a sentence or paragraph before executing it. After using the ".R" command to place the sentence or paragraph into the sentence stack, the ".A" (append), ".C" (change), or ".I" (insert) commands may be used to modify the VOC item in the sentence stack, and the ".X" (execute) command to execute it or the ".S" (save) command to save the modified version in the VOC file.

.R#	Recall sentence number "#" from the PERFORM sentence stack as sentence number 1.
.R name	Load the contents of the VOC file SENTENCE or PARAGRAPH whose id is "name" into the PERFORM sentence stack.

FIGURE 1. Formats of the RECALL sentence stack command.

```
:.L
03 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
02 LIST ORGANIZATION EMPLOYEE.ID
01 LIST EMPLOYEES BY DEPARTMENT

:.R3
SSELECT ORGANIZATION WITH DEPARTMENT = SALES

:.L
04 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
03 LIST ORGANIZATION EMPLOYEE.ID
02 LIST EMPLOYEES BY DEPARTMENT
01 SSELECT ORGANIZATION WITH DEPARTMENT = SALES

:.R LISTS
1 line(s) loaded from VOC file.

:.L
05 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
04 LIST ORGANIZATION EMPLOYEE.ID
03 LIST EMPLOYEES BY DEPARTMENT
02 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
01 SORT VOC WITH TYPE = "S" NAME DESC F2 ID.SUP DBL.SPC HEADING "SENTEN
CES stored in your VOCABULARY file"
```

FIGURE 2. Examples of the RECALL sentence stack command.

7.9 .S (Save Sentence/Paragraph From PERFORM Sentence Stack)

Any sentence or group of sentences in the PERFORM sentence stack may be saved in the VOC file as a stored system SENTENCE or PARAGRAPH by the ".S" sentence stack command.

Any sentence in the PERFORM sentence stack may be saved as a stored system sentence in the VOC file with the ".S" (save) sentence stack command. Additionally, any group of sentences in the PERFORM sentence stack may be saved in the VOC file as a stored system PARAGRAPH with the ".S" sentence stack command. The format of the ".S" sentence stack command is:

```
    .S name s# e#
```

where "name" is to be the name of an item in the VOC file whose type will be either "S" (for SENTENCE) or "PA" (for PARAGRAPH); "s#" is the number of the sentence to be saved, or the number of the first sentence of the sentence stack to be saved as a paragraph; and "e#" is the number of the last sentence of the sentence stack to be saved as a paragraph in the VOC file.

If the ".S" sentence stack command is used with the parameters "s#" and "e#" omitted, PERFORM will save sentence number one from the sentence stack as a stored system sentence in the VOC file. If the ".S" sentence stack command is supplied, and the "e#" parameter omitted, PERFORM will save sentence number "s#" from the sentence stack as a stored system sentence in the VOC file.

If the ".S" sentence stack command includes both "s#" and "e#" parameters, then the "e#" parameter must be numerically less than the "s#" parameter. With this form of the ".S" command, PERFORM will save the group of sentences which begins with sentence number "s#" and ends with sentence number "e#" (and which includes both of these sentences) as a stored system paragraph in the VOC file. This paragraph will have sentence number "s#" as its first sentence, and sentence number "e#" as its last sentence.

```
     .S name s# e#
```

FIGURE 1. Format of the SAVE sentence stack command.

```
:.L

08 LOGIN
07 SELECT STUDENTS WITH EVERY GRADE > 3.5
06 RUN ADMIN HONORS.LIST
05 SELECT STUDENTS WITH EVERY GRADE < 2.0
04 RUN ADMIN PROBATION.LIST
03 SSELECT STUDENTS BY ZIP.CODE
02 RUN ADMIN GRADE.REPORTS
01 LIST STUDENTS BY SCHOOL BY MAJOR BY.DSND GRADE.POINT.AVERAGE LPTR

:.S GRADE.REPORTS 7 2
:.L GRADE.REPORTS

PA saved on 11:30:58 12 JUN 1980 by ADMIN
SELECT STUDENTS WITH EVERY GRADE > 3.5
RUN ADMIN HONORS.LIST
SELECT STUDENTS WITH EVERY GRADE < 2.0
RUN ADMIN PROBATION.LIST
SSELECT STUDENTS BY ZIP.CODE
RUN ADMIN GRADE.REPORTS

7 lines listed from VOC file.

:.S GRADE.AVERAGE.REPORT
:.L GRADE.AVERAGE.REPORT

S saved on 11:30:24 12 JUN 1980 by ADMIN
LIST STUDENTS BY SCHOOL BY MAJOR BY.DSND GRADE.POINT.AVERAGE LPTR

2 lines listed from VOC file.
```

FIGURE 2. Examples of the SAVE sentence stack command.

7.1Ø .X (Execute Sentence In PERFORM Sentence Stack)

Any sentence in the PERFORM sentence stack may be re-executed by the ".X" stack command.

Any sentence in the PERFORM sentence stack may be re-executed by the ".X" sentence stack command. The format of the ".X" sentence stack command is:

```
.X#
```

where "#" is the number of the sentence in the stack which is to be re-executed. This number may be from 1 to 99 or, if omitted, will be taken as 1 (the default value).

.X#	Re-execute sentence number "#" from the PERFORM sentence stack.

FIGURE 1. Format of the EXECUTE sentence stack command.

```
:.L
03 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
02 LIST ORGANIZATION EMPLOYEE.ID
01 LIST EMPLOYEES BY DEPARTMENT

:.X3
SSELECT ORGANIZATION WITH DEPARTMENT = SALES

:.L
04 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
03 LIST ORGANIZATION EMPLOYEE.ID
02 LIST EMPLOYEES BY DEPARTMENT
01 SSELECT ORGANIZATION WITH DEPARTMENT = SALES

:.X2
LIST EMPLOYEES BY DEPARTMENT

:.L
05 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
04 LIST ORGANIZATION EMPLOYEE.ID
03 LIST EMPLOYEES BY DEPARTMENT
02 SSELECT ORGANIZATION WITH DEPARTMENT = SALES
01 LIST EMPLOYEES BY DEPARTMENT

:.X
01 LIST EMPLOYEES BY DEPARTMENT
```

FIGURE 2. Examples of the EXECUTE sentence stack command.

7.11 ? (Terminate PERFORM Sentence Which Contains Error)

If you make an error while typing a sentence at the PERFORM ":" prompt, you may type a "?" at the end of the sentence to direct PERFORM to put the sentence on the sentence stack, but not to attempt to execute it.

Often when typing a complicated sentence from the keyboard of your terminal, you may make an error which you do not notice until you are nearly finished with the sentence. When this occurs, it can be extremely frustrating to have to "kill" the sentence and begin again from the beginning, particularly if you are unfortunate enough to make a different error on a subsequent attempt!

To relieve this frustration, PERFORM will recognize a question mark typed at the end of a sentence as a signal that the sentence should not be executed, but should be retained on the PERFORM sentence stack. Therefore, when you detect an error in a sentence you are typing, you may simply terminate the sentence with a question mark, then use the sentence stack ".C" command to correct the error you noticed, and the ".X" command to execute the corrected sentence.

There is no "format" to the "?" sentence stack command, but an illustration of some typical errors might serve to demonstrate its usefulness.

```
:LIST EMPLOYEES BY SALART BY DEPARTMENT NAME SALARY DEPARTMENT?
:.C/ART/ARY/
LIST EMPLOYEES BY SALARY BY DEPARTMENT NAME SALARY DEPARTMENT
:.X

:LIST CUSTOMERS BY TYPE BY LOC?
:.C/C/DICT C/
LIST DICT CUSTOMERS BY TYPE BY LOC
:.X
```

Any sentence which needs changing before execution?

FIGURE 1. Format of the OOPS sentence stack command.

```
:LIST STUDWNTS WITH COURSE = CS301 AND GPA > 3.5?
:.C/W/E/
LIST STUDENTS WITH COURSE = CS301 AND GPA > 3.5
:.X
```

FIGURE 2. Example of the OOPS sentence stack command.

7.12 PERFORM Sentence Stack Retention (STACKWRITE)

PERFORM may be directed to save your sentence stack even after you leave the PRIME INFORMATION environment. A special item in the VOC file, named STACKWRITE, is used to specify whether PERFORM should or should not retain your sentence stack.

The PERFORM sentence stack is always active when you are actually in the PRIME INFORMATION environment (i.e., when you are actually using PRIME INFORMATION). However, when you leave the PRIME INFORMATION environment, your sentence stack is not normally retained. If you wish for PERFORM to retain your sentence stack so that it will be available to you the next time you use PRIME INFORMATION, you can specify this with the special VOC file item named STACKWRITE.

The VOC file item STACKWRITE is an "X" item (to denote a user-defined VOC file item type); this item type is described in detail in the section of this manual titled "USER Record Definition". The STACKWRITE item makes use of Field Two of the VOC file record, to specify whether the retention of the PERFORM sentence stack is to be turned ON or OFF. Specifically, Field One of the VOC file item named STACKWRITE contains an X (to identify the item as a user record), and Field Two contains either the word ON (to specify that the PERFORM sentence stack should be retained after you leave the PRIME INFORMATION environment), or the word OFF (to specify that the PERFORM sentence stack may be erased when you leave the PRIME INFORMATION environment).

When you specify that the PERFORM sentence stack is to be retained, it is kept as a record in the file named &SAVEDLISTS&. The name of the record is comprised of the special characters "&&S", your LOGIN account name, and your terminal port number, with a period separating each element. Thus, for a user whose LOGIN account name is JONES, using a terminal on port number 3, and who has an item named STACKWRITE in the VOC file of the account where he first used PRIME INFORMATION specifying "ON" in field two, there would be a record in the file &SAVEDLISTS& named &&S.JONES.3.

Note that, in order to re-activate a PERFORM sentence stack saved earlier, it is necessary that <u>all</u> of the following events occur:

1. You LOGIN to the same account as before.

2. You use the same terminal as before.

3. You enter the same PRIME INFORMATION account as you last used.

4. The STACKWRITE item in the VOC file has "ON" in Field Two.

```
 id:  STACKWRITE

001:  X If field two = "OFF", stack is not kept after logoff.
002:  ON
```

FIGURE 1. The VOC file item STACKWRITE.

8 APPENDIX A: SPECIAL FILES

8.1 &ED& File

The special file named &ED& is used by the EDITOR to store command files which may be used during the editing process.

The PRIME INFORMATION EDITOR is able to work with commands which are stored in a file, as well as commands which are typed at a terminal. For example, it is possible to save a specified range of commands which have been executed in an editing session by telling the EDITOR to take commands from the EDITOR's command stack and save them in a named file (using the EDITOR's ".S" command). Additionally, a pre-stored set of EDITOR commands may be loaded into the EDITOR's command stack by using the ".R" command. Still other options allow you to execute (with the ".X" command) a file of pre-stored EDITOR commands, to list (with the ".L" command) the contents of an EDITOR command file, or to delete (with the ".D" command) an EDITOR command file.

With all of these EDITOR command file operations, you may (optionally) specify the name of the PRIME INFORMATION file where the commands are stored, in addition to the name of the record. However, if you do not specify a PRIME INFORMATION file name, the EDITOR will use a default file name of &ED&. If the operation being executed is a SAVE (using the ".S" EDITOR command), and the &ED& file does not exist, the EDITOR will create this file for you automatically.

Each record in the &ED& file must have in Field One the letter E, to identify the record as containing executable EDITOR commands. Fields Two through the end of the record must each contain a valid EDITOR command.

The records in the &ED& file may be built from within the EDITOR (by using the ".S" command, after placing the appropriate commands on the EDITOR's command stack), or may be built by using the EDITOR directly on the records in the &ED& file. The &ED& file should be used only to store EDITOR command records, for the sake of consistency.

For additional information concerning the EDITOR and its pre-stored command file operations, see THE PRIME INFORMATION EDITOR REFERENCE, IDR3904.

```
:ED &ED& EXAMPLE
11 lines long.

----: P
0001: Ē Pre-stored Command saved at 15:17:28 25 JUN 1980 by WENDY
0002: L COM
0003: <
0004: F *
0005: -1
0006: >
0007: DROP
0008: Y
0009: I  INSERT I_COMMON.DEF
0010: T
0011: C/I = I + 1/I += 1/9999
Bottom at line 11
```

FIGURE 1. Example of a record in &ED& file.

8.2 &HOLD& File

The &HOLD& file is used to "hold" spooled reports which were created by the PRIME INFORMATION software after using the SETPTR command's Mode 3.

The &HOLD& file is a Type 1 (sub-UFD) file to which reports or printed output are directed (rather than to SPOOLQ) after a user has executed the SETPTR command, specifying Mode 3. This Mode of the SETPTR command is explained in detail in the section of this manual titled "SETPTR Command".

If the &HOLD& file does not exist in the account at the time the SETPTR command with the Mode 3 option is executed, then it will be created automatically by the SETPTR processor. When the &HOLD& file is created, its dictionary (named D_&HOLD&) is also created. This dictionary contains a Type "X" entry with the name 'NEXT.HOLD', that is used as a counter by the SETPTR command processor. If the appropriate Mode 3 option is specified (see the description of the SETPTR command), the SETPTR processor will increment this counter and append it to the output report name; this method is used to provide unique or successive report names. "NEXT.HOLD" is initialized to 1 when the dictionary to the &HOLD& file is created, and is incremented to 9999 before being reset back to 1.

After a report has been sent to the &HOLD& file, it is then possible to edit, delete, or spool the report. Although it is logically a report, it is actually a record in a standard PRIME INFORMATION file, and is subject to any process that is appropriate for a record in a Type 1 file. The report will continue to reside in the &HOLD& file until it is explicitly deleted, or until another report with the same name is written to the file.

The &HOLD& file must remain as a local file to each account: it is not possible to send reports to remote &HOLD& files with the SETPTR command.

8.3 &MAP& File

The special file named &MAP& is used by the MAKE.MAP.FILE command to create an INFORM-accessible copy of the PRIME INFORMATION system CATALOG space.

The catalog space of a PRIME INFORMATION system contains object programs in executable form, with those programs having been placed into the catalog space by using the CATALOG command (described in this manual). When a program is placed into the system catalog space, information relevant to that program is stored with it. Some of the components of that information include catalog name, the date it was cataloged, the account name from which it was cataloged, etc.

This relevant information is directly available by using the MAP command (described in this manual), but only in the order in which each program is encountered in the catalog space. In order to apply the full power of INFORM (the high-level on-line query and report generator supplied as part of each PRIME INFORMATION system) to the system catalog, there is a command named MAKE.MAP.FILE, which will create an INFORMATION data file from the current catalog contents. (See the description of the MAKE.MAP.FILE command elsewhere in this manual.) The file created by this command, for later use by INFORM, is named &MAP&.

The &MAP& file is a standard PRIME INFORMATION data file, described by a dictionary associated with it (D_&MAP&). Once this file is created, reports may be generated from it using the full power of INFORM, including selection criteria and sequencing criteria. The &MAP& file is normally defined only in the ISYS account, and other accounts may have a REMOTE pointer to this file. If you wish to have your own private &MAP& file, you must create it within your account, and copy the dictionary from the DICT of the ISYS &MAP& file, before using the MAKE.MAP.FILE command.

Note that since the &MAP& file is created only upon request, its contents become out-of-date with the first use of any command which manipulates catalog space (such as CATALOG, or DELETE.CATALOG). If you wish to use the &MAP& file to create reports to aid in system catalog management, you should use the MAKE.MAP.FILE command immediately before creating any such reports. The &MAP& file is not updated automatically by any of the catalog management commands, but is simply recreated (at your request, with the MAKE.MAP.FILE command) from current system catalog contents.

```
:LIST DICT &MAP&
LIST DICT &MAP& 12:18:28  10-27-80  PAGE     1
FIELD NAME.... TYPE LOCATION....... CONV DISPLAY NAME... FORMAT SM

@ID             D   0                    &MAP&           4L     S
NAME            D   1                                    28L    S
TYPE            D   2                                    2R     S
ARG             D   3                                    3R     S
REF             D   4                    REF             5R     S
                                         COUNT
OBJ             D   5                    OBJ             5R     S
                                         BYTES
XREF            D   6                    XREF            5R     S
                                         BYTES
SYMB            D   7                    SYMB            5R     S
                                         BYTES
WHO             D   8                                    7L     S
DATE            D   9               D2/                  8R     S
@               PH  ID.SUP NAME ARG
                    REF OBJ XREF
                    SYMB WHO DATE

11 records listed.
```

FIGURE 1. The dictionary of the &MAP& file.

```
:LIST &MAP& BY.DSND OBJ

LIST &MAP& BY.DSND OBJ 09:32:24  10-28-80  PAGE     1
                           REF   OBJ  XREF  SYMB
NAME.................. ARG COUNT BYTES BYTES BYTES WHO.... DATE....

-LIST                   2    36 33032  4346  4426 ADMIN   10/07/80
-ED                     2    27 29278  3358  2762 ADMIN   10/07/80
-ENTROC                 2     0 15460  1976  1768 ADMIN   10/07/80
-ENTRO                  2     0 12312  1542  1570 ADMIN   10/07/80
-FUNCTION               8     0  8558  1740  1126 ADMIN   10/07/80
-CLEAN.ACCOUNT          2     0  4042   494   824 ADMIN   10/24/80
 . . .
-CREATEF                2     0  2390   228   408 ADMIN   10/07/80
-COPY                   2     1  2232   350   730 ADMIN   10/07/80
-GROUP.STAT             2     0  2176   324   992 ADMIN   10/07/80
-T.READ                 2     0  1820   182   518 ADMIN   10/07/80
-FORMAT.SUB             2     6  1816   270   224 ADMIN   10/07/80
-ENTRO.ASSOC           18     0  1800   226   396 ADMIN   10/07/80
Press <NEW LINE> to continue...Q
```

FIGURE 2. An INFORM listing of the &MAP& file.

8.4 &PH& File

The special file named &PH& is used by the PHANTOM command to store the COMO output files from PHANTOM processes started within the account.

When the PHANTOM command is used to start a background process within a PRIME INFORMATION account, the PHANTOM processor automatically creates a COMO file which becomes a log of the total activity of the PHANTOM process. This COMO file is stored as a record in the PRIME INFORMATION file named &PH&.

Each record in the &PH& file is named according to the following format:

```
O_phantom.name_ttttt
```

The "O_" portion of this record name follows the PRIMOS convention of naming COMO files with this prefix, so that they may be readily identified as this type of file.

The phantom.name portion of this record name is taken from the command line at the time the PHANTOM command is invoked. It is actually the first word of the process to be executed as a phantom task. For example, the command:

```
:PHANTOM LIST VOC NO.PAGE
```

would produce a COMO record in the &PH& file named O_LIST_ttttt.

The ttttt portion of this record name is derived from the time of day when the phantom process was begun. It is derived by the INFO/BASIC function INT(TIME()), to produce an integer that is unique to that instant of time. With this suffix to the record name, these COMO files may be kept unique within an account, even if several users all initiate the same phantom process at nearly the same time.

The &PH& file is a Type 1 file, and is created automatically by the PHANTOM command if there is no &PH& file in the account when the PHANTOM command is invoked. Since it is a Type 1 file, the items in this file (which appear as records to PRIME INFORMATION users) are actually PRIMOS files within a sub-UFD. Therefore, these items may be accessed at either the PRIME INFORMATION level or at the PRIMOS level.

These COMO files are created by the PHANTOM processor so that the correct operation of a phantom process may be verified after the process has completed. The user may edit the COMO file that goes with the phantom task of interest, or may SPOOL any of these files to obtain a printed copy. It is the user's responsiblity to delete these COMO files from the &PH& file when they are no longer needed.

See the PHANTOM command description in this manual, and the COMO command description in this manual and in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, for additional information.

```
     O_phantom.name_ttttt

where:

    O_              - Standard PRIMOS prefix for COMO files.

    phantom.name    - First word of phantom process command.

    ttttt           - A Unique number derived from time of day.
```

FIGURE 1. Format of record names in &PH& file.

```
:LIST &PH&

RECORD NAME....................

O_BENCH_69030
O_BENCH.SORT_56182
O_BENCH.SORT_36478
O_BENCH.201_53327
O_BENCH.2001_47969
O_LIST_53994

 6 records listed.
```

FIGURE 2. Examples of record names in &PH& file.

8.5 &SAVEDLISTS& File

The file named &SAVEDLISTS& is used by the SAVE.LIST, GET.LIST, and DELETE.LIST commands, to store the lists which these commands use.

When the SAVE.LIST command is used to save a list of record identifiers (a list created by the SELECT or SSELECT commands), the list is saved in a PRIME INFORMATION file named &SAVEDLISTS&. Each record in the &SAVEDLISTS& file is named according to a convention described in detail in the sections of this manual titled "SAVE.LIST Command", "GET.LIST Command", and "DELETE.LIST Command". Briefly, this convention is that "list.name" will be used if specified with the SAVE.LIST command, or &TEMPport#& (where "port#" is the user's port number) will be used if no "list.name" is specified. Either of these names will have a three-digit sequence number appended to it, from 000 to 999, by the SAVE.LIST, GET.LIST, or DELETE.LIST commands, as needed to keep the saved lists to a manageable size (5000 records or fewer).

Most lists will contain few enough record identifiers to allow a single record to store all of them; in this case, the record representing such a list will be named either "list.name000" or "&TEMPport#&000". If the number of record identifiers in the list is greater than 5000, then the list will be broken into records of 5000 items each. These records will be consecutively numbered: e.g., "list.name001", "list.name002", "list.name003", etc.; or "&TEMPport#&001", "&TEMPport#&002", etc.

The &SAVEDLISTS& file is a Type 1 file, and is created automatically by the SAVE.LIST command if there is no &SAVEDLISTS& file in the account when this command is invoked. Since it is a Type 1 file, the items in this file (which are records to PRIME INFORMATION users) are actually PRIMOS files within a sub-UFD; therefore, these items may be accessed at either the PRIME INFORMATION level or at the PRIMOS level.

Note that when the PRIME INFORMATION editor (ED) is used to modify the records in the &SAVEDLISTS& file, it is necessary for the user to append the three-digit sequence number to the saved list name. For example, if a select list were saved with the command:

```
    :SAVE.LIST LATE.PAYMENTS
```

then the command to edit this saved list would be:

```
    :ED &SAVEDLISTS& LATE.PAYMENTS000
```

If the saved list contained more than 5000 records, it would be necessary to edit LATE.PAYMENTS001, LATE.PAYMENTS002, etc.

USER-SPECIFIED list.name	DEFAULT LIST NAME &TEMPport#&
list.name001	&TEMPport#&001
list.name002	&TEMPport#&002
list.name003	&TEMPport#&003
list.name004	&TEMPport#&004
list.name005	&TEMPport#&005
list.name006	&TEMPport#&006
list.nameseq	&TEMPport#&seq

FIGURE 1. Format of record names in &SAVEDLISTS& file.

```
:LIST &SAVEDLISTS&

RECORD NAME.....................

STATES000
CUST000
ORDERS000
&TEMP3&000
MAIL.LIST001
MAIL.LIST002
MAIL.LIST003
MAIL.LIST004
&TEMP7&000
&TEMP11&000
```

FIGURE 2. Examples of record names in &SAVEDLISTS& file.

8.6 &TEMP& File

The special file named &TEMP& is used by the CLEAN.ACCOUNT command to store items which were found in the account, but were not usable by PRIME INFORMATION. Additionally, the process which updates your account to a new release level may store items from your VOC file in the &TEMP& file, if those items are inconsistent with new items that have the same name.

The file named &TEMP& is used as a temporary file by the PRIME INFORMATION system. The principal purpose of this file is to serve as a temporary repository for items which are encountered in your account, but which either are not fully (or correctly) defined, or which conflict with items that have the same name but are of different type. In either case, PRIME INFORMATION will store such items in the &TEMP& file, rather than simply deleting them, so that you may examine these items and determine what should be done with them.

One process which is likely to store items into the &TEMP& file is the CLEAN.ACCOUNT processor, which is described elsewhere in this manual. If CLEAN.ACCOUNT finds an item in your VOC file which is a file definition record, but the file that it defines is not in your account, the VOC file item will be moved to the &TEMP& file, and deleted from the VOC file.

Another process which uses the &TEMP& file is the process which updates your account to the latest Release level. Since a new Release may include some new features which are defined in the VOC file, it is possible that the names of these new features will conflict with items already in your VOC file, items that you created. For example, if you had a paragraph named RESIZE on an early Release of PRIME INFORMATION, and you updated your account to Release 4, the update processor would move your paragraph named RESIZE to the &TEMP& file, and replace it in the VOC file with the verb named RESIZE.

The &TEMP& file is a Type 1 file, and is created automatically by any command which requires it, if there is no &TEMP& file in the account when that command is invoked. Since it is a Type 1 file, the items in this file (which appear as records to PRIME INFORMATION users) are actually PRIMOS files within a sub-UFD. Therefore, these items may be accessed at either the PRIME INFORMATION level or at the PRIMOS level.

8.7 &UFD& File

The special file named &UFD& is a synonym used by PERFORM for the PRIMOS UFD which is the equivalent of the PRIME INFORMATION account.

PRIMOS views a PRIME INFORMATION account as a collection of files and sub-UFD's in a PRIMOS UFD. This same PRIME INFORMATION account is viewed by PERFORM as a collection of required files (such as VOC, VOCLIB, user-defined data files, etc.). In order to support PERFORM's view of the account, it is occasionally necessary to access PRIMOS-level files or sub-UFD's within the account UFD, even though these file types and sub-UFD's are not (strictly speaking) formally defined to PRIME INFORMATION.

For this purpose, there is a PRIME INFORMATION file named &UFD&, which is a PERFORM-accessible file conceptually, but is actually the PRIMOS-level UFD where the PRIME INFORMATION account is located.

Examples of commands which use the &UFD& file include COMO (which uses &UFD& to search for files whose name begins with C_), and CLEAN.ACCOUNT (which uses &UFD& to search for files whose name is not defined in the VOC file for that account).

8.8 APP.PROGS File

The APP.PROGS file in the master ISYS account is used as a mechanism for distributing programs which are not supported by PRIME.

The APP.PROGS file is a Type 1 (sub-UFD) file residing in the master account named ISYS. This file is used to store the source and object code to programs which are not supported by PRIME, and whose correct operation is not warranted by PRIME. These programs have been contributed by PRIME INFORMATION users, and PRIME is acting as a distribution arm for these contributed programs. In some cases the programs in the APP.PROGS file will have been written by employees of PRIME, while in other cases the programs were contributed by one of PRIME's authorized distributors of PRIME INFORMATION systems, or by customers who have purchased PRIME INFORMATION systems. No matter what their origin may have been, they all belong to the category of unsupported programs.

In some cases, the people at PRIME who have direct responsibility for the PRIME INFORMATION product will be familiar with the workings of a particular program in the APP.PROGS file. More often, the programs in this file will have been placed there without even a cursory examination by anyone at PRIME. The intent behind this free distribution of contributed programs is that valuable programs which may happen to have no commercial value, may still be made available to PRIME INFORMATION customers on a "use them at your own risk" basis. The programs in the APP.PROGS file should be read carefully and understood well before even attempting to use them, lest you inadvertantly destroy some critical portion of your files by uninformed use. Indeed, perhaps the best use of these programs is either as a model of how other persons utilize INFO/BASIC, to give you new insights into how (or how not) to best use this language; or as a template that you can use to create a program by finishing it to your organization's standards.

As the number of customers of PRIME INFORMATION systems increases, it is possible that the number of programs in the APP.PROGS file will become quite large. You should examine this file (each time you receive a new copy of it) carefully, and extract those programs that you wish to have on your system. Having done this, you probably should delete the APP.PROGS file (using the DELETE.FILE command), in order to reclaim the disk space that this file would otherwise occupy. Be sure you have a backup copy of the APP.PROGS file before you delete it from the system, in case you discover that there is a program in it which precisely solves a problem whose solution you are about to have to invent!

8.9 DICT.DICT File

The special file named DICT.DICT is the master dictionary which describes the contents of every other dictionary on a PRIME INFORMATION system.

PRIME INFORMATION systems are completely dictionary-driven: every file used with the PRIME INFORMATION software is described by an associated dictionary. Since each dictionary is itself a PRIME INFORMATION file, and since each dictionary is available for use with INFORM, there must be a dictionary which describes the contents of these special files. That "master" dictionary is named DICT.DICT.

Every PRIME INFORMATION system is supplied with a file named DICT.DICT, located in the ISYS account. The DICT.DICT file is the master dictionary which describes the contents of every other dictionary on the PRIME INFORMATION system.

The DICT.DICT file is defined only in the ISYS account, and other accounts have a REMOTE pointer to this file. The contents of the DICT.DICT file should not be modified by anyone, since they document the contents and format of every dictionary used on the entire PRIME INFORMATION system.

```
:LIST DICT.DICT

LIST DICT.DICT 16:29:Ø2  1Ø-3Ø-8Ø  PAGE      1
FIELD NAME.. TYPE LOCATION....... CONV DISPLAY NAME... FORMAT SM ASSOC

@ID          D    Ø                    FIELD NAME      14T    S
ID           D    Ø                                    15L    S
FIELD        D    Ø                                    15L    S
KEYR         D    Ø                                    15R    S
KEY          D    Ø                                    15L    S
ITEM.ID      D    Ø                                    1ØT    S
CODE         D    1                                    4L     S
F1           D    1                                    4L     S
VERB         D    2                                    6ØT    S
FUNC         D    2                                    6ØT    S
LOC.L        D    2                    LOCATION        4ØT    S
KEY.CODE     D    2                                    8R     S
EXP          D    2                    FIELD # or      3ØT    S
                                       EXPRESSION
DATA.FILE    D    2                    DATA.FILE.NAME  3ØT    S
MACRO        D    2                                    6ØT    S
A2           D    2                                    3R     S
LOC          D    2                    LOCATION        15T    S
F2           D    2                                    8T     S
CONV         D    3                                    4L     S
DICT.FILE    D    3                    DICTIONARY.NAME 3ØT    S
F3           D    3                                    1ØT    S
NAME         D    4                    DISPLAY NAME    15T    S
F4           D    4                                    12L    S
FORMAT       D    5                                    6L     S
F5           D    5               FORM                 12L    S
                                  AT
SM           D    6                                    1L     S
F6           D    6                                    1ØT
S/M          D    6                                    1L     S
ASSOC.NAME   D    7                                    25L    S
ASSOC        D    7                                    1ØL    S
F7           D    7                                    1ØT    S
GROUP        D    7                                    25L    S
F8           D    8                                    1ØT    S
F9           D    9                                    2T     S
F1Ø          D    1Ø                                   4T     S
F11          D    11                                   1ØT
F12          D    12                                   1ØL    S
F13          D    13                                   1ØT
F14          D    14                                   1ØT
F15          D    15                                   1ØT
F16          D    16                                   1ØT
ARGS         D    16                                   4R     M
F17          D    17                                   1ØT
F18          D    18                                   1ØT
F19          D    19                                   1ØT
F2Ø          D    2Ø                                   1ØT
F21          D    21                                   1ØT
```

```
F22            D     22                                     10T
F23            D     23                                     10T
DESC           I     FIELD(F1,'                             30T    S
                     ',2,9999)
SOURCE         I     IF F1=I THEN F2      SOURCE FUNCTION 20T
                     ELSE IF F1='D'
                     THEN 'FIELD#
                     ':F2 ELSE
                     'MACRO ':F2
UCOPY          I     @ID:CHAR(254):S                        78L    S
                     TR('-',LEN(@ID)
                     ):CHAR(254):CHA
                     R(254):@RECORD:
                     STR(CHAR(254):'
                     ',3)
TYPE           I     IF F1[1,1]='P'                         2L##   S
                     THEN F1[1,2]
                     ELSE F1[1,1]
DISP.WIDTH     I     IF LEN(F4) GE   MD0                    8R     S
                     LEN(FMT('
                     ',F5)) THEN
                     LEN(F4)+1 ELSE
                     LEN(FMT('
                     ',F5))+1
FIELD.NO       I     IF NUM(F2) THEN                        5R     S
                     F2 ELSE 999999
@LPTR          PH    BY TYPE BY
                     KEY.CODE TYPE
                     LOC CONV NAME
                     FORMAT SM ASSOC
@              PH    BY TYPE BY
                     KEY.CODE TYPE
                     LOC CONV NAME
                     FORMAT SM ASSOC
@DESC          PH    BY TYPE BY LOC
                     TYPE DESC LOC
                     CONV NAME
                     FORMAT SM
ALL.F          PH    F1 F2 F3 F4 F5
                     F6 F7 F8 F9 F10
                     F11 F12 F13 F14
                     F15 F16 F17 F18
                     F19 F20 F21 F22
                     F23
```

FIGURE 1. The contents of the DICT.DICT file.

8.10 ISYS Account

The ISYS account on a PRIME INFORMATION system is the central account, serving as a repository for several files which are accessible from every other account on the system.

Every PRIME INFORMATION system has an account named ISYS (for Information SYStem), which serves as the "master" account for the system. The ISYS account is pivotal to the operation of PRIME INFORMATION, and additionally contains several files which are used by every other account on the system (via remote file pointers in each account's VOC file).

The ISYS account is the only PRIME INFORMATION account supplied by PRIME with every system. Because of its pivotal nature to the operation of the system, in general it should be left unchanged. However, some of the files in this account might be specially tailored to the needs of your organization, by your System Administrator. For example, the file named NEWACC resides in the ISYS account. This file can be thought of as a "master" VOC file, since every new account created as a PRIME INFORMATION account has its VOC file created from the NEWACC file. Your System Administrator may wish to remove some entries from the NEWACC file (such as CREATE.FILE or DELETE.FILE), and to add some others which are processes created by your organization.

Other files which you will find in the ISYS account include:

(1) APP.PROGS: This is a file which contains source code to programs which are available for use on PRIME INFORMATION systems, but which are not supported by PRIME. These programs should be viewed as models which can be custom-modified for your use, rather than as finished programs.

(2) BLTRS: This file contains records used by the BLOCK.PRINT and BLOCK.TERM commands, to produce block letters on the system printer or on your terminal.

(3) DICT.DICT: This is the master dictionary which describes the contents of every dictionary on the system. In the same way that the DICT portion of a data file describes the contents of that data file, the DICT.DICT file is the DICT portion of all dictionaries on the system. Its contents should not be modified in any way (no changes, additions, or deletions should be done).

(4) ENTRO.DISCUSSIONS: This file is used by ENTRO to provide narrative descriptions during data entry operations using ENTRO.

(5) ENTRO.PROCESSES: This file is used by ENTRO to define some of the processes required during data entry and verification.

(6) ISYS.MENU.FILE: This file is used to store MENU records that are to be available to all accounts on the system. Each account can have its own menu file(s) for "private" menus, but the ISYS.MENU.FILE can be used to make a menu available to everyone.

(7) ISYS.VOCLIB: This file is a globally-accessible file where your System Administrator can store processes (such as Paragraphs) which are to be available to everyone on the system.

(8) SYS.HELP: This file contains a record for each standard system component (Verbs, Paragraphs, Sentences, Files, Keywords, etc.), for use by the HELP command. Essentially, this file constitutes documentation available on-line.

(9) VERBS: This file contains the object code of every command supplied with the PRIME INFORMATION system. This object code is already cataloged, but can be re-cataloged by the PRIMOS-level command file named C_BOOT.INFO; however, only your System Administrator should do this, since the catalog space is completely cleared by this command file. (Note that the object code for PERFORM is re-shared each time the system is cold-started, and therefore must always be available in ISYS>VERBS.)

8.11 NEWACC File

The NEWACC file is the "master" copy of the system VOC file, from which every new account's VOC file is created and, for new Releases, updated.

When a PRIME INFORMATION account is created, a "standard" vocabulary file (the VOC file), which defines all of the components of the PRIME INFORMATION system that are common to every account, must be created for the new account. Once this standard VOC file is created (by copying the contents of the NEWACC file into it), the account's VOC file can be custom-tailored to its intended use, by adding and/or deleting elements of the VOC file that are particular to this account. For further explanation of how a PRIME INFORMATION account is created, see the sections in this manual on the INFO command (a PRIMOS-level command which can transform a standard PRIMOS UFD into a PRIME INFORMATION account), and the LOGTO command (a PRIME INFORMATION-level command which also can transform a standard PRIMOS UFD into a PRIME INFORMATION account).

The NEWACC file resides in the master ISYS account, described elsewhere in this manual. This file contains an entry for every component of the PRIME INFORMATION software as delivered by PRIME, provided that all components in the NEWACC file are usable by (and intended for) "general-use" accounts. (Specifically, certain commands are not in the NEWACC file, because they are intended for use only by the System Administrator and should not be copied into every VOC file in every account.)

Every system can, to some degree, have a custom-tailored NEWACC file, so that each account created on the system is built to a known standard. The System Administrator may wish to remove some of the PRIME-supplied commands from the NEWACC file (such as DELETE.FILE, DELETE.CATALOG, CLEAR.LOCKS, etc.) in order to restrict these operations from general availability. The System Administrator may additionally wish to add some commands to the NEWACC file, for processors which are supplied not by PRIME, but by the particular installation.

After an account's VOC file has been created from the NEWACC file, there is no further need (within that account) for the NEWACC file unless and until the PRIME INFORMATION software is upgraded to a later Release level. When PRIME supplies a new version of PRIME INFORMATION, and it is installed on your system, all accounts which already existed before the new Release will be out-of-date. Each time you enter such an account (either via the INFO command from PRIMOS, or via the LOGTO command from within PRIME INFORMATION), you will be informed that your account requires updating to the current Release level, and will be given an opportunity (by the INFO command only) to have this updating done for you. If you affirm that your account should be updated to the current Release level, the updating is accomplished by copying the contents of NEWACC into your VOC file. Since there may be new features of PRIME INFORMATION in any new Release, and since the names of these features may duplicate names that you have created in your own VOC file, this update process is careful to preserve for you entries that are different in character from the item with the same name in the NEWACC file. (See the description of the &TEMP& file for additional information on this process.)

8.12 PHANTOM Command Files

When the PHANTOM command is invoked within PERFORM, it creates a PRIMOS COMI file in the PRIME INFORMATION account which is used to control the phantom task.

When the PHANTOM command is used to start a background process within a PRIME INFORMATION account, the PHANTOM processor automatically creates a COMI file which controls the total activity of the PHANTOM task. This COMI file is stored in the PRIME INFORMATION account (PRIMOS UFD) from which the PHANTOM command was invoked. The COMI file is named according to the following format:

```
C_PHANTOMn_ttttt
```

The "C_" portion of this record name follows the PRIMOS convention of naming COMI files with this prefix, so that they may be readily identified as command input files.

The PHANTOMn portion of this record name is the constant PHANTOM catenated with the user number of the task which invoked the PHANTOM command (the same user number displayed by the WHO command).

The ttttt portion of this record name is derived from the time of day when the phantom process was begun. It is derived by the INFO/BASIC function INT(TIME()), to produce an integer that is unique to that instant of time. With this suffix to the record name, these COMI files may be kept unique within an account, even if several users all initiate the same phantom process at nearly the same time.

These COMI files may be inspected by the user to verify the sequence of operations which was specified (on behalf of the phantom task) to the PERFORM processor. The user may edit the COMI file that goes with the phantom task of interest, or may SPOOL any of these files to obtain a printed copy.

See the PHANTOM command description in this manual, and the COMI command description in this manual and in the PRIMOS COMMANDS REFERENCE GUIDE, FDR3108, for additional information.

```
     C_PHANTOMn_ttttt
```

where:

C_	- Standard PRIMOS prefix for COMI files.
PHANTOMn	- PHANTOM catenated with user number.
ttttt	- A unique number derived from time of day.

FIGURE 1. Format of PHANTOM command input file names.

```
C_PHANTOM2_53244
C_PHANTOM2_40365
C_PHANTOM5_37178
C_PHANTOM3_15324
C_PHANTOM8_42762
C_PHANTOM2_37153
```

FIGURE 2. Examples of PHANTOM COMI file names.

```
* COMI FILE GENERATED BY PHANTOM VERB
A
ATTACH &PH& 1/1
COMO O_KWIC.BUILD_50776_4671 -NTTY
ATTACH
INFO -PHANTOM
KWIC.BUILD
QUIT
ED
DELETE C_PHANTOM4_50776_4671
LOGOUT

FILE C_&PH&
CO C_&PH&
COMO -END
LOGOUT
```

FIGURE 3. Example of a PHANTOM COMI file.

8.13 SYS.HELP File

The special file named SYS.HELP contains a succint description of every VOC file item supplied with a PRIME INFORMATION system.

Every PRIME INFORMATION system is supplied with a file named SYS.HELP, located in the ISYS account. The SYS.HELP file contains a record for every VOC file item supplied as part of a standard PRIME INFORMATION system. As an example, in the case of commands, each record comprises a succint description of the command, including such information as the order and type of parameters used with the command; a brief description of its purpose; some examples of its use; and, where appropriate, its interaction with other commands on the system.

The SYS.HELP file is used directly by the HELP command (described in this manual). When you type "HELP VOC.item.name", the information which is displayed as a result (describing "VOC.item.name") was retrieved from the SYS.HELP file.

Additionally, the SYS.HELP file may be used with INFORM, to produce written documentation sorted by command name, or by keyword name, etc. The SYS.HELP file is a standard PRIME INFORMATION data file, described by a dictionary associated with it (D_SYS.HELP). Using this file and its dictionary, reports may be generated using the full power of INFORM, including selection criteria and sequencing criteria.

The SYS.HELP file is defined only in the ISYS account, and other accounts have a REMOTE pointer to this file. If you wish to have your own private HELP file, describing VOC file items which you have added to the standard group supplied with the system, you may do so with the USER.HELP file, described elsewhere in this manual. The contents of the SYS.HELP file should not be modified by anyone except your System Administrator, since they document the operation and use of standard components of a PRIME INFORMATION system as supplied by PRIME.

```
:LIST DICT SYS.HELP

LIST DICT SYS.HELP 10:06:31  10-28-80  PAGE     1
FIELD NAME.... TYPE LOCATION....... CONV DISPLAY NAME.. FORMAT SM ASSOC

@ID             D    0                   Processor      10T    S
TEXT            D    1                   Description    65T    M
FULL.TEXT       I    @RECORD             Description    65T    S
@LPTR           PH   BY @ID TEXT
                     DBL.SPC
@               PH   TEXT DBL.SPC
TITLE           PH   TEXT DBL.SPC
@ENTRO          PH   TEXT

 7 records listed.
```

FIGURE 1. The dictionary of the SYS.HELP file.

```
:SSELECT VOC BY TYPE

320 records selected.

:LIST SYS.HELP BREAK.SUP "'P'" @ID FULL.TEXT ID.SUP COL.SUP LPTR
```

FIGURE 2. Producing an INFORM report from the SYS.HELP file.

8.14 USER.HELP File

The special file named USER.HELP is similar in purpose to the SYS.HELP file, except that the contents of the USER.HELP file are defined and controlled by your installation.

Every PRIME INFORMATION system is supplied with a file named SYS.HELP, located in the ISYS account. The SYS.HELP file contains a record for every VOC file item supplied as part of a standard PRIME INFORMATION system. The SYS.HELP file is used directly by the HELP command (described in this manual). When you type "HELP VOC.item.name", the information which is displayed as a result (describing "VOC.item.name") was retrieved from the SYS.HELP file.

The USER.HELP file is conceptually similar in purpose and use to the SYS.HELP file, and its format is identical. The USER.HELP file is intended to contain descriptive records for every element of the VOC file that has been created at your installation. When someone types "HELP VOC.item.name", the item "VOC.item.name" is searched for first in the USER.HELP file (if there is a USER.HELP file in the account), and if encountered, is displayed. Whether or not "VOC.item.name" was found in the USER.HELP file, it is next searched for and (if found) displayed from the SYS.HELP file.

If you create a USER.HELP file for an account, you should use the dictionary of the SYS.HELP file to describe the USER.HELP file, and you should follow exactly the format of the records in the SYS.HELP file. If you do not do so, the HELP command may not work correctly on your system. The content of the USER.HELP file is completely under your control, but its format must match the format of the SYS.HELP file, in order for the HELP command to work properly.

```
:LIST DICT USER.HELP

LIST DICT USER.HELP 10:06:31  10-28-80  PAGE     1
FIELD NAME.... TYPE LOCATION....... CONV DISPLAY NAME.. FORMAT SM ASSOC

@ID            D    0                    Processor      10T    S
TEXT           D    1                    Description    65T    M
FULL.TEXT      I    @RECORD              Description    65T    S
@LPTR          PH   BY @ID TEXT
                    DBL.SPC
@              PH   TEXT DBL.SPC
TITLE          PH   TEXT DBL.SPC
@ENTRO         PH   TEXT

 7 records listed.
```

FIGURE 1. The dictionary of the USER.HELP file.

```
:SSELECT VOC WITH TYPE = 'V' AND F3 = 'IB' BY TYPE

71 records selected.

:LIST USER.HELP BREAK.SUP "'P'" @ID FULL.TEXT ID.SUP COL.SUP LPTR
```

FIGURE 2. Producing an INFORM report from the USER.HELP file.

8.15 VERBS File

The VERBS file is one of the files contained in the system master ISYS account, and is used to store the object code of all of the PERFORM commands supplied by PRIME with the PRIME INFORMATION system.

The PRIME INFORMATION file named "VERBS" is one of the pivotal files supplied by PRIME with every PRIME INFORMATION system. The VERBS file resides in the special account named ISYS (described elsewhere in this manual), and is used to store the object code of every PERFORM command supplied by PRIME.

Most of the commands documented in this manual (and in the THE INFORM REFERENCE GUIDE, IDR3905 and THE PRIME INFORMATION EDITOR REFERENCE, IDR3904 manuals) are implemented by command processors written in INFO/BASIC. After these programs have been compiled, their object code is stored in the VERBS file of the ISYS account. This object code is also cataloged on each system before delivery, so that the programs are actually executed from the system catalog space. However, the system catalog space can be re-created, if necessary, by re-cataloging all of the object code in the VERBS file. Within the ISYS account, there is a PRIMOS-level command file, "C_BOOT.INFO", which will accomplish this re-cataloging automatically. Only your System Administrator should ever use this command file, because it will be necessary to re-catalog all programs which were written at your installation after using it (the catalog space is completely cleared as part of this procedure).

One other use for the VERBS file is to re-share PERFORM itself, each time the system is cold-started. During cold-start, the bootstrap process will load the object code whose name is "$PERFORM" from the VERBS file, into shared system space. While it would be possible to delete all of the other object code programs in the VERBS file and continue to use PRIME INFORMATION, it is necessary to retain at least the file named "$PERFORM" because of this cold-start process.

The VERBS file is reserved for the exclusive use of PRIME, as part of its mechanism for delivering the software that constitutes PRIME INFORMATION systems. The contents of this file may not be altered in any way, nor may new programs be placed into this file, without voiding your installation's entitlement to support of your system. If any of the PRIME-supplied commands are modified, replaced, or augmented, PRIME will no longer be responsible for the correct operation of your PRIME INFORMATION system.

Index

PERFORM INDEX