AskPlus

DATA REPORTING

Reference Manual

Version E.7.8

VITAL Soft INCORPORATED

www.vital-soft.com

PMB 501
26910 92nd Ave NW - C5
Stanwood, WA 98292-5437
800-848-2576 * Fax (360) 654-8170

Copyright Notice

AskPlus Reference Manual

Version E.7.8 – April, 2006

Conventions used in this manual

Throughout this manual a standard notation will be used to express the syntax of commands and statements. These conventions are as follows:

UPPER-CASE	Literal symbols to be used as they appear. `FIND expression`
italics	Variable fields to be filled in by the user. These fields often have a definition in the glossary.

`COURIER`	An example normally appears in COURIER. Example:
	`>FIND CUST-NO = ""`
	`What is the value of CUST-NO 1234`

<< >>	When text appears between these double GT and LT characters (angle brackets), it is considered a comment.
CTRL	This key-word preceding a character indicates a Control character. The Control key and the character must be pressed simultaneously. Example: CTRL Y
[ ]	An element between brackets is optional. Several elements stacked inside a pair of brackets means the user may select any one or none of these elements.
{ }	When several elements are stacked within braces the user must select one of them.

```
FORM [[[ domain.] set.] item ]
      [[ domain.] set        ]
```

```
INSEL { filename }
      {  *       }
```

|	has the same effect as braces if the options are each reduced to a single symbol. `[ LEFT | RIGHT | CENTER ]`
`...`	A horizontal ellipsis indicates that the previous bracketed element may be repeated. `[, parm] [...]`

Table of Contents

1 Introduction

- Overview
- Entering Commands
- Data Handled by AskPlus

1.1 Overview

AskPlus is a comprehensive reporting solution that provides advanced query, report and update capabilities coupled with excellent performance. AskPlus was originally developed for HP 3000 (MPE) systems to query and report from Image and TurboImage databases and files. Over the years, AskPlus has been enhanced to run on a variety of UNIX systems to query and report from Eloquence and Oracle databases.

AskPlus is a command driven language. This manual documents the syntax of each of the AskPlus commands and statements and explains how each of the commands should be used. Visimage for Windows is a report writing solution that works together with AskPlus. Using Visimage, users can create reports without learning the AskPlus language. For more information on Visimage contact Vital Soft, or visit our web site at www.vital-soft.com.

Using this Manual

This manual has been designed as a reference manual. It is organized into the following chapters:

Chapter 1 - ***Introduction*** **...** (Page 1)

This chapter provides information about the structure of the AskPlus language and the basic data objects that may be included in a report.

Chapter 2 - ***Working with Data*** **...** (Page 19)

Before developing a report it is necessary to define the data sources that will be used. This chapter discusses the commands used to access different data sources.

Chapter 3 - Selecting Data ...

The first step in report writing is selecting the data. This chapter gives examples and explains the concepts of data selection. The FIND command is discussed in detail. Once selected, the data can be used to print reports, create files or update data fields.

Chapter 4 - ***Reporting Data*** **... (Page 81)**

This chapter discusses the REPORT command which is used to create a formatted report from the selected data.

Chapter 5 - ***Creating Data Files*** **... (Page 135)**

This chapter discusses the SAVE command which is used to create data files from the selected data.

Chapter 6 - ***Updating Data*** **... (Page 153)**

This chapter discusses the different commands that can be used to update or delete the selected data records.

Chapter 7 - ***Utility Commands*** **... (Page 167)**

This chapter discusses all other commands which have not been discussed in the previous chapters.

Chapter 8 - ***Understanding Expressions*** **... (Page 201)**

This chapter discusses the rules for creating AskPlus expressions. It discusses the different operands and operators, system variables and special operators.

Chapter 9 - ***Configuring AskPlus*** **... (Page 243)**

This chapter provides general configuration information.

Basic Language Structure

The AskPlus language is composed of commands and statements. Some commands are very simple; others are more complex and can contain one or more statements.

Commands

The syntax of each individual command is discussed later in this manual. Each command has the following general format:

```
command-name  [ statement [;] ]  [...]
```

The following rules are common to all commands.

- The semi-colon character ';' indicates the end of a statement. However, this character is optional in those statements whose structure is simple enough for AskPlus to recognize the end of the statement.
- The end of a line is assumed to be the end of a statement if the statement can reasonably be terminated there. Otherwise, AskPlus expects the rest of the statement to continue on the next line.
- If the character '&' is entered as the last character in the line, the statement will be continued on the next line. No *blank* is generated between the two lines. This character is not considered a separator when it appears at the end of the line.
- Certain commands (REPORT and SAVE) **must** be terminated by the END

statement. The FIND command may also use this statement as a terminator but it is not mandatory; the FIND command ends logically when a condition has been entered and when this condition is syntactically complete.

Statements

The syntax of each statement is described in the command(s) in which it is used. Some statements are common to several commands. The general format is as follows:

```
statement-name  [ statement-text [;] ]
```

Each statement is generally composed of one or more of the following elements:

keyword	Keywords are special reserved words that must be entered as shown. Keywords can generally be abbreviated. Keywords are only recognized when they are entered in the correct location. The same keyword may have a different meaning (or no special meaning) depending on how it is used.
identifier	Symbol composed of a string of up to 20 characters. It is sometimes abbreviated as *ident* in this manual. The characters allowed in an identifier depend on the context. The characters can be alphabetic, numeric or, in most cases, one of the following: + - * / ? ! ' # % & @ $ _. There is no distinction between upper and lower case characters. The first character of an identifier **must** be alphabetic.
separator	The characters used as separators in AskPlus are the following: `" , . : ; ( ) =` and the *space or blank* as well as the end of line. The number of blanks is not significant.
constant	Character string which may or may not be surrounded by quotes. If quotes are used, all the characters entered are recognized as is; lower case being different from upper case, blanks are significant and separators (other than quotes) become regular characters. To place quotes in a character string, you must enter two ("This is a ""valid constant"""). If the constant is not surrounded by quotes, the string is up shifted and must not contain separators or any special characters.

Expressions

Many AskPlus statements allow the use of expressions. AskPlus supports arithmetic, string and conditional expressions involving data fields, temporary variables and constants. AskPlus expressions are discussed in more detail in ***Chapter 8, Expressions & Formats*** (Page 201).

1.2 Entering Commands

AskPlus can be used in several different ways:

interactive	By entering the commands one after another on your terminal. When AskPlus is waiting for a new command, it displays the '>' prompt followed by the cursor. When it is waiting for another statement to be entered for the same command, '>>' is displayed. See Chapter 9 for instructions on starting AskPlus in interactive mode on your system.
semi-interactive	By executing a predefined sequence of commands stored in an *Xeq* file. See below for more information.
batch mode	By launching an offline job or script that contains all of the necessary commands to be executed. The entire job/script is stored in a file and executed without any interaction from a user.
Visimage	By designing a report using Visimage, Visimage will execute your report using AskPlus commands.

In some cases AskPlus behaves differently depending on whether or not it is working in interactive mode:

▪ If an error occurs and AskPlus is working in:

interactive	The command/statement is rejected and the user enters a new command or statement.
semi-interactive	The command/statement is rejected and the next line of the file is read.
batch mode	By default AskPlus terminates with a "fatal error". However, you can prevent this using the ASSIGN BATCH command.

▪ If additional information is required and AskPlus is working in:

interactive	AskPlus will prompt the user for the missing information on the standard input file (the terminal).
semi-interactive	When prompting for user supplied values (?parm or ""), AskPlus will prompt the user for the missing information on the standard input file (the terminal). In all other cases, default values will be used.

batch mode	AskPlus will complete the missing information using the best available default values.

XEQ and Include Files

Semi-interactive mode uses 'XEQ' files. A *batch process* can also use them. These are simple text files containing a sequence of commands. An *XEQ* file is started by using the ***XEQ command*** (Page 198). An *XEQ* file may call another XEQ file and when the end of a file is encountered, the calling file resumes execution in sequence. When the last *XEQ* file is finished, AskPlus returns control to the standard input file, that is, the terminal in session mode or the *job file* in *batch* mode. There is no limit to the number of nested *XEQ* files. However, AskPlus cannot go beyond the capabilities of the operating system.

When an *XEQ* file is running, the commands and statements are echoed to the standard output file (terminal in *session* mode and printer in *batch* mode) unless the QUIET option has been requested.

INCLUDE files are similar to *XEQ* files and work the same way. They contain a sequence of statements and are called using the INCLUDE statement available in the main commands. Statements are not echoed unless they are called within an *XEQ* file.

Continuation Character - &

When a command or statement must be continued on the next line, use the '&' to signify that the command (statement) will continue on the next line. If the command or statement is not logically complete, the continuation character is not required and AskPlus will expect the command (statement) to continue on the next line.

Examples:

```
FIND SHARE-FILE.SUFFIX = 0 AND SHARE-FILE.REST-FLAG-11
<> 80, 81 AND
SHARE-FILE.BALANCE > 0
```

The continuation character is not required since AskPlus knows that something must follow the logical AND.

```
FIND SHARE-FILE.SUFFIX = 0 AND SHARE-FILE.REST-FLAG-11
<> 80, 81 &
AND SHARE-FILE.BALANCE > 0
```

In this is example the continuation character **is required** since the find condition is logically complete at the end of the line..

Macro Instructions

AskPlus allows commonly repeated expressions (or statements) to be defined as a macro. When used, the name of a macro must be preceded by the '%' character. Macros can be defined with or without parameters. When using parameters, the exact number of parameters defined must be specified. A macro can only be used in a context where it makes sense.

Example:

```
>MACRO AVERAGE(X,Y) = ((X + Y) DIVR 2)
>FIND %AVERAGE(ITEM1, ITEM2) <  100
```

The MACRO command is used to define new macro elements or to redefine existing ones. See the ***MACRO*** (Page 186) and the ***ASSIGN MACROFILE*** (Page 171) commands (Chapter 7, Utility Commands) for more information on using macros. Additional information is also available in the *Visimage Administrator Manual.*

User Break

The task in progress can be interrupted by simultaneously pressing the **CTRL Y** keys (*control* and Y). A break affects each task differently depending on the command or the phase in which it occurs. In all cases, a message appears indicating that a break request has been received. Generally speaking, the current command is interrupted and the user is prompted to enter another command.

When a FIND command is being executed, the break behaves in a special manner. AskPlus displays the number of entries read and selected, then asks you whether you want to continue the search. If you do not want to continue, the search stops and the entries already qualified are selected.

REDO (MPE Only)

If you want to re-enter the previous line, or correct it, just type CTRL B <return>. The last line is redisplayed and you can correct it using all the terminal's editing keys. When the new command is ready, just type *return*. Remember that the entire line, including the characters to the right of the cursor, is executed. Caution: this feature is only available on Hewlett-Packard terminals or compatibles which support *block mode* or block transmission.

Yes/No Responses

In interactive mode under certain circumstances, AskPlus will prompt you for a 'yes or no' answer. Regardless of the language being used during the session, answers can be given in any of the different languages recognized, in either upper or lower case. The recognized answers are:

YES	NO	English
Y	N	English abbreviations
OUI	NON	French
JA	NEIN	German
SI	NO	Spanish/Italian

User Defined Prompts

In many cases it is desirable to enter a user defined value each time a report is executed. User defined prompts can be requested using the null value ("") or using the "?prompt" syntax. If a user defined prompt is requested in interactive or semi-interactive mode, the user is prompted to enter the actual value to be used in the command.

Examples:

- Example using a null value:

```
>FIND ITEM1 = "" OR ...
What is the value of ITEM1
>>20
```

- Example using ?prompt:

```
>FIND (ITEM2 / 2 * 15) + 100 > ?MIN-AMOUNT OR ...
What is the value of MIN-AMOUNT
>>50000
```

Qualifying Sets & Items

The same set name may be used in various databases (or various domains). Similarly, the same item name may be used several times in the same domain and even in different domains. If, in an expression, the name of a set or an item is ambiguous, and if you are in interactive mode, AskPlus will display the list of domains to which the set belongs or the list of sets to which the item belongs and prompt you to choose which one you want to use (by entering the number).

Example:

```
>FIND CUST-NO = 1000
CUST-NO is a member of these sets
1 BASEA.M-CUSTOMER
2 BASEA.D-ORDER
Which one do wish to use? 2
```

The item selected is BASEA.D-ORDER.CUST-NO.

If you are working in a non-interactive mode, AskPlus automatically uses the first one in the list and tells you it is doing so by displaying a message:

```
BASEA.M-CUSTOMER used
```

To avoid the risk of ambiguity, fully qualify the name of the item::

```
FIND BASEA.D-ORDER.PROD-NO = 1000
```

You can also place sets in a priority list which AskPlus will use to qualify ambiguous names (See the DATA-SETS command).

Compatibility Between Ask Version C and AskPlus

AskPlus is able to execute reports created with Ask Version C. The command "ASSIGN SYNTAX ASKC" allows you to run old procedures and jobs with practically no modification. This option can also be set up by entering the following Jcw command :

```
SETJCW ASKC=1
```

Proc-files can be used in AskPlus. A procedure stored in a *Proc-file* is automatically run in ASKC mode. After execution, AskPlus is switched back to the previous mode.

In ASKC mode, some features of AskPlus are unavailable. Every time a procedure is being modified, it is advisable to translate it to AskPlus syntax.

ASKC select files are readable by AskPlus. You can also, if necessary, generate ASKC select-files with AskPlus. To create ASKC select files, you must set the system variable ASKSEL. For example:

```
SETJCW ASKSEL=1
```

1.3 Data Handled by AskPlus

AskPlus is capable of reporting data from a wide variety of data sources. These data sources can be structured databases containing datasets (tables) and items (columns) or they can be individual files with or without an inherent structure. Before describing the commands used to access data, it is important to understand the terminology that AskPlus uses when working with databases, datasets (tables) and files. The three main types of objects are the *domain*, *set* and *item*:

Domain

A domain is a generic term used by AskPlus to refer to a data source. A domain is either a **database** or a **file**. The following is a list of most of the different domains that are supported by AskPlus:

- Image and TurboImage databases (MPE Only)
- Eloquence databases (UNIX & NT)
- Oracle databases (UNIX & NT)
- Fixed Length files
- KSAM files with fixed-length records (MPE Only)
- AskPlus (Portable) SD files
- MPE SD files
- KSAM SD files (MPE Only)
- SUPRTOOL SD files (MPE and UNIX)
- Powerhouse subfiles (MPE Only)
- BRW subfiles (MPE Only)
- UNIX byte stream ASCII files
- UNIX binary files with a specified file layout

In AskPlus, once a domain has been opened, the domain is referred to by a short name. There can be only one domain open with the same name. A domain must be opened before it can be used. The following commands are used to handle domains:

OPEN	Used to open a domain and to add it to the list of opened domains. The list of open domains is known as the current environment.
DATA-BASE	Does the same thing as OPEN, but only for a database.
DEFINE	Used to open a domain and chain various commands defining the environment.
SUPERDEFINE	Used to check and modify open domains and open others

	if required.
CLOSE	Used to close a domain.
DATA-SET	Used to specify a priority list of data sets to use when locating items.
FORM	Display information about the structure of a domain.
ENVIRONMENT	Similar to the OPEN command, this command is used to open a domain from an environment file. See the Visimage Administrator Manual for more information on using environment files.

Each of these commands are discussed in more detail in ***Chapter 2, Accessing Data*** (Page 19).

Set

A structured database is made up of a collection of data sets or tables. In AskPlus, the term Set is used to refer to a data set (TurboImage / Eloquence) or a table (Oracle). A set has a symbolic name in the form of an identifier which must be unique in each domain. The syntax of a set is as follows:

```
[domain.]set
```

In non-structured domains, in other words domains which are not organized as *databases* or provided with a dictionary (*KSAM* and *MPE*), the set name is the same as the domain name. Thus the fully qualified set name becomes "domain.domain". On MPE systems, you can replace the domain name by the keywords MPE and KSAM.

Examples:

- `LABELS.LABELS`
- `MPE.LABELS`
- `ZIPCODES.ZIPCODES`
- `KSAM.ZIPCODES`

Fully qualifying a set name is how you can distinguish between sets with the same name in different domains.

Item

An *item* (also known as a column in Oracle databases) is the name of a single element (part) of an entry in one or more sets. An *item* has a symbolic name in the form of an identifier, a type and a length (expressed by the term *typelen* in this manual). An *item* can be an array, in which case, in order to distinguish between the members of the array, the name of the item is followed by a subscript in parentheses representing the *index* number. The complete syntax for an item is:

```
[[domain.]set.]item[(subscript)]
```

Object

The basic operand handled in AskPlus expressions is called an *object*. An object may be any one of the following:

```
{field              }
{set                }    << FIND cmd only >>
{variable           }
{system-variable    }
{constant           }
```

Where:

field	Is an item (or a portion of an item) in any open set.
set	In the FIND command, the name of a set may be used as an object.
variable	Is a temporary variable which exists only for the duration of an AskPlus session or command. Variables are also called registers.
system-variable	Represents a special constant such as today's date or the current page number in a report.
constant	A numeric or string constant.

Field: Item or Sub-item

In its simplest form a field is the name of an item. A field may also be a portion of an item, known as a sub-item. To specify a sub-item, you must specify a byte offset followed by the type and length. Finally, the field definition may also contain an access path (or link name) that determines how the field should be accessed from the primary. The general format of a *field* is as follows:

```
[link..][[domain.]set.]item[(idx)][.sub-field]
```

or

```
[link..][domain.]file.sub-field     << non-structured file only >>
```

Where:

link	is the name of the link used to access the specified set. See the ***LINK statement*** (Page 62) for information on creating links. The link name is optional.
domain	is the name of the domain. The domain name is optional.
set	is the name of the set. The set name is optional.
item	is the name of the item. Except in the special case shown above, the item name is required.
idx	is the subscript (or index) of a compound or array item. If no subscript is specified, the first element of a compound item will be used. In certain situations, the '@' may be used as the subscript, to indicate that all elements of the compound item should be used.
sub-field	redefines a zone within the item (or file). The format of a sub-field is as follows: `(offset:typelen)`
offset	is the byte (character) position of the sub-field in the item (or file). The first character is byte position number 1.
typelen	defines the type and length of the field.

Examples:

- `M-CUSTOMER.CUST-NAME`
- `ORDER-DATE.(5:X2)`
  << extract the month from Order-Date >>
- `LINK1..ZIPCODES.(7:20)`
  << extract a text field from a non-structured file >>

A *field* has a type and length. If it contains an item name, the *field*'s default type and length is that of the item. However, if it is followed by a *sub-field*, the type and length are defined by the *sub-field* regardless of the item's type. This is a redefinition of the data, no conversion is done.

Variable

A variable is a temporary field which is created dynamically by the user and which exists only for the duration of the session or the command. It is used to hold a value or an array of values of any type in order to perform any type of computation. The general format is as follows:

```
[#]name[(idx)][.sub-field]
```

Where:

name	Is the name of the variable (register).
idx	Is the subscript (or index) of an *array* variable. If no subscript is specified, the first element of the array item will be used. In certain situations, the '@' may be used as the subscript, to indicate that all elements of the array should be used. If the variable is not an array, no index is used.
sub-field	Redefines a zone within the item (or file). The format of a sub-field is shown above.

Variables are created using either a ***NEWREG*** *command (Page 181)* or a NEWREG *statement*. If the NEWREG command is used, the variable is global and therefore exists for the entire AskPlus session (or until it is deleted using the ***PURGEREG command*** (Page 183)). It can also be reinitialized using the ***LOAD command*** (Page 182). Example:

```
NEWREG REG:3p12 = "12345"
 ...
LOAD REG(2) = "23456"
...
PURGEREG REG
```

Variables can also be created locally within the main commands (FIND, REPORT, SAVE) using a NEWREG *statement.* In this case the variable is considered local and exists only for the duration of the command. A local variable which is used but has not been explicitly defined is created automatically by AskPlus. If the variable is used in an arithmetic operation, depending on whether the context is scientific or commercial, the variable is created in floating format type (double precision) or as a large-capacity integer type. If the variable is used in a logical operation, it is created as a boolean (or logical) type. Otherwise it is created as a text variable.

A variable has a symbolic name (which must follow the rules for an identifier) which is normally preceded by the '#' character to distinguish it from another type of object. A variable also has a type and length and can be a dimensioned (array). Example:

```
#REG1(2)
```

See Chapter 4, Reporting Data for more information on the NEWREG statement and working with variables. See also ***Chapter 7, Utility Commands*** (Page 167) for more information on the NEWREG, LOAD and PURGEREG commands.

System Variables

AskPlus has a variety of system variables (also called special objects) that are used to provide the value of variables like the current date and time or the current page number in a report.

A system variable is designated by its name which is normally preceded by the '$' character to avoid confusion with an item name or variable name. System variables vary in type and length according to the information they return. The general format is as follows:

```
[$]name[.sub-field]
```

Where *sub-field* has the same syntax as shown above for a field.

Examples:

$DATE8	The current date in CCYY/MM/DD format.
$PAGENO	The current page number of the report.
$SECONDS	The current date and time in internal format. This value is used in many date calculations as today's date.

See ***Chapter 8, Expressions & Formats*** (Page 232) for a detailed list of all the AskPlus system variables.

Constant

The general format for constants is as follows:

```
{"string"[(S[MART])]          }
{nn("string")                 }
{number                       }
{user-prompt                  }
```

Where:

string	Represents a character string enclosed in "double quotes". A string that does not contain any blanks or special characters does not normally require quotes.
(S[MART])	The SMART keyword indicates that the string contains wildcard characters as described below.
nn	To specify a constant which is a long string of the same character,

	you can add a repetition factor (nn) in front of the constant and place the constant in parentheses. This format may only be used in the REPORT command and is not valid in a FIND.
number	Is a number. A number may be enclosed in "double quotes".
user-prompt	Is a prompt for a constant value to be entered by the user.

Examples:

- `"this is a constant"`
- `123`
- `123.45`
- `"-1"`
- `SMITH`
- `WA`
- `"WA@"(SMART)`
- `132("*")`
- `""                  << null value - user prompt >>`
- `?prompt`

Null Values and User Prompts

If a null value ("") or a user prompt (identifier preceded by '?') is found in an AskPlus command and if the user is in interactive mode, the user will be prompted for a value to be assigned to each parameter. Then, the command is executed by using the values entered. In batch mode, the value(s) is read from the job file. If, the prompt is executed from inside an XEQ file (in batch mode), the value(s) will be read from the next lines of the job/script which follows the XEQ command, not from the XEQ file itself.

Example of Prompt in Batch using an XEQ File:

In the following example, assume that the XEQ file contains two separate user prompts. The first is for a date, the second is an update code. The example is shown using UNIX syntax, but is equally valid on other platforms.

```
/ASKPLUS/askplus -b << EOD_ASK
open database, password;
```

```
xeq ../PATH/XFILENAME
20051001
UPDATE

exit
EOD_ASK
```

The value "20051001" will be used for the first prompt and the value "UPDATE" will be used for the second prompt. In most cases, the values of the date prompts will be supplied using a variable syntax supported by your batch processing software, for example UC4 or Maestro.

Wildcard Characters

Wildcard characters are used to make approximate matches when comparing **string** values. *Wildcard* characters only have a meaning when used in logical conditions which make comparisons (with operators such as *equal to* (=), *not equal to* (<>), etc.). For a constant to be considered to contain a wildcard character, the (SMART) attribute must be added, or the SEARCH keyword must be specified as part of the FIND command.

?	matches a single alphabetical character
#	matches a single numerical character
\	matches any single character
@	matches zero or more characters

Examples:

- `FIND D-ORDER.DATE = "##08##"(S)`

  Selects all the entries in D-ORDER for which the item DATE contains "08" as the 3rd and 4th characters, the other characters being any numerical characters. If the data format is "DDMMYY" or "YYMMDD", the month of August will be selected.

- `FIND SEARCH M-CUSTOMER.CUST-NAME = "@JOHN@"`

  Selects all the M-CUSTOMER entries which contain "JOHN" anywhere in the customer name.

See ***Wildcard (Smart) Constants*** (Page 212) for more information.

Special Characters

In AskPlus, certain characters have a special meaning when they are used as the first character in the name of a field. These characters are used by AskPlus to signify what type of field is being specified. The characters are as follows:

Character	**Meaning**
#	Variable (Register)
$	System Variable
!	Special Operator
%	Macro
&	Continuation Character

2 Accessing Data

- Overview
- OPEN Command
- CLOSE Command
- DATA-SET Command
- FORM Command
- Working with Files

2.1 Overview

AskPlus is able to query (select) and report data from a wide variety of data sources. In this chapter we look at the commands that are used to grant access to these data sources. AskPlus supports both **databases** and **files.** On the HP 3000, AskPlus supports Image and TurboImage databases. On UNIX systems, AskPlus supports Eloquence and Oracle. This chapter discusses the following commands:

OPEN	Used to open (grant access to) all of the different data sources supported by AskPlus. The syntax for other commands (BASE, DATA-BASE, DEFINE and SUPERDEFINE) will also be discussed.
CLOSE	Used to CLOSE (stop accessing) a specific data source.
DATA-SET	Used to specify a list of priority sets that are used to qualify ambiguous item names.
FORM	Used to view the structure of a data source.

This chapter also includes a discussion of the different types of files supported by AskPlus and methods for describing the layout of these files. It will be helpful to understand the following terms:

Domain	A generic term that refers to any data source supported by AskPlus. A domain is either a database or a file.
	If the syntax of a command refers to a *domain*, then the command applies to **both** databases and files. The syntax will use *base* or *database* if the command is limited to a database and will use *file* or *filename* if the command is limited to a file.
Environment	The environment is the collection of domains that are open and accessible to the user.

2.2 OPEN Command [OP]

This command is used to open a domain (database or file) and add it to the current environment. The syntax of this command varies slightly depending on the type of domain being accessed.

Opening Databases (TurboImage & Eloquence) and Files:

When accessing a TurboImage database, Eloquence database or a file, use:

```
OPEN domain [,password [,mode]] [= local-name ] [,USING
dictionary ] [;]
```

This command will open the domain using the specified password and mode. If a local name is specified, the domain will be known by that name. The name of a dictionary may be specified using the USING clause. See below for opening an Oracle database.

Where:

domain	Is the name of the database or file to be opened. The domain name may refer to a database (TurboImage or Eloquence) or to a file. The syntax of the domain name depends on the host system. On HP 3000 systems, domain names should be specified using the "file[.group[.account]]" syntax and not using POSIX syntax. The domain name **may** be case sensitive depending on the host system.
password	Is the password that specifies the level of access to be granted to the current user. Passwords are case sensitive on most platforms. When updating data with AskPlus, you must specify a password with the appropriate access rights. When accessing a database, a password is normally required. If no password is specified, creator access is assumed. On HP 3000 systems, use the password to specify the lockword for a file that has been password protected. See below for a special syntax (HP 3000 only) that allows the password to be specified as part of the domain name.
mode	Is the mode to be used when opening the database. For TurboImage and Eloquence, the mode is a number between 1 and 8. If no mode is specified, mode 5 (read/share) is assumed. The most common modes are 1 (write/share), 3 (write/exclusive) and 5 (read/share). For more information on modes, refer to the appropriate database documentation.
local-name	When a domain is opened, it is known by it's short name. For

	example, the domain "base.group.acct" will be referred to as "base". The local name parameter allows you to specify a different short name that will be used to identify the domain. The local name is optional, however, it is necessary to use a local name if you want to open more than one domain with the same short name.
dictionary	On HP 3000 systems, you can specify the name of a dictionary to use when accessing this domain. The dictionary must be specified using the "assign scope" command. See the *Visimage Administrator Manual* for more information on creating dictionaries.
;	The semi-colon terminates the command. Normally, if no password or mode are specified, and you are working interactively, AskPlus will prompt the user to enter the appropriate password and/or mode. However, if the command is terminated by a semi-colon, there will be no prompt.

Domain Syntax:

On HP 3000 systems, a special syntax allows the password to be embedded as part of the domain name. When this syntax is used, the mode can be specified as the next parameter without an additional comma as a place holder.

```
name/password[.group[.account]]
```

A special syntax is used to open an Eloquence database on a remote system. This syntax is not valid on HP 3000 systems:

```
[machine or ip_addr]/dbname
```

HP 3000 Examples:

```
>OPEN base1/MANAGER.demo.account,5;
```

- Opens the database "BASE1.DEMO.ACCOUNT" with the password "MANAGER" in mode 5.
- The domain name may be entered in upper or lower case.

```
>OP histra = histry
Password  =>>
Open mode =>>
```

- Opens "HISTRA", in the current group and account with the local name "HISTRY".
- Prompts for a password and mode.

```
>OP file.group;
```

- Opens the file "FILE.GROUP", with no password or mode specified.
- There is no prompt for a password because the command was terminated with a semi-colon.

UNIX Examples:

```
>OPEN membrs, pswd;
```

- Opens the Eloquence database "MEMBRS" with the password "pswd" in mode 5.
- Eloquence database names are not case sensitive.

```
>OP ../dir1/dir2/fname;
```

- Opens the file "../dir1/dir2/fname" with no password or mode.
- There is no prompt, because the command was terminated with a semi-colon.

```
>OP 123.123.123.123/dbname,WRITEPASS,1;
```

- Opens the database "dbname" with the password "WRITEPASS" in mode 1 (write/share).
- The database is opened on a remote system with the specified ip addr.
- If you have problems connecting to an Eloquence database, make sure that the system variable EQ_DBSERVER is correctly set. See the Eloquence documentation for more information.

Opening Oracle Databases:

When accessing an Oracle database, use:

```
OPEN ORACLE username [,password] [;]
```

Where:

ORACLE	is a keyword that indicates we are opening an Oracle database.
username	is the Oracle username that will be used to open the database.
password	is the password for the specified user.

When working with Oracle, the following Oracle system variables must be set:

ORACLE_HOME	Specifies the Oracle installation directory.

ORACLE_SID	Specifies the Oracle instance to be used.
ASK_TWO_TASK	Only needed if the Oracle database is located on a remote server.

Example:

```
open oracle sales, password;
```

- The database will be opened using the username "sales" using the specified password. There is no mode parameter when opening an Oracle database.

Discussion:

- Domains can also be opened via a configuration file using the ENVIRONMENT command. See the *Visimage Administrator Manual* for more information.
- If you do not specify a password with the base name, it is requested automatically after the command is entered.
- If you do not enter the mode immediately after entering the data-base command, the system prompts you for it. You can answer by simply typing a carriage return, in which case the default mode value is used: mode 5 (shared read).
- No prompt is made for the password or mode if the command is terminated by a semi-colon.
- You can optionally specify a local name when opening a domain if the actual name of the domain conflicts with the name of a domain which has already been opened. In this case, the local name instead of the actual name must be used whenever referring to the domain.
- There is no limit to the number of open commands that may be specified to open multiple domains.
- The same domain may be opened more than once, provided it is given a unique local name.

Note See also the commands: `DATA-BASE`, `DEFINE`, `SUPERDEFINE`, `CLOSE`.

DATA-BASE Command [B]

Open a TurboImage or Eloquence database only. Cannot be used to open files or other databases.

```
{DA[TA-BASE]} [=] database [= local-name] [;]
{B[ASE]     }
```

Where database has the syntax:

```
dbname [,password [,mode]]
```

The DATA-BASE (BASE) command is used to open an Image or Eloquence database. It cannot be used to open a file. This command is a sub-set of the OPEN command. It provides backwards compatibility with QUERY and previous versions of Ask.

Where:

dbname	is the name of the database to be opened.
password	is the password that specifies the level of access to be granted to the current user.
mode	is the mode to be used when opening the database.
local-name	the local name to use for this database.
;	The semi-colon terminates the command. Normally, if no password or mode are specified while working interactively, AskPlus will prompt the user to enter the appropriate password and/or mode. However, if the command is terminated by a semi-colon, there will be no prompt.

See the OPEN command for more information about each of the above parameters.

Database name Syntax:

On HP 3000 systems, a special syntax allows the password to be embedded as part of the database name. When this syntax is used, the mode can be specified as the next parameter without an additional comma as a place holder.

```
dbname/password[.group[.account]]
```

A special syntax is available to open an Eloquence database on a remote system. This syntax is not valid on HP 3000 systems:

```
[machine or ip_addr]/dbname
```

HP 3000 Examples:

```
>B=base1/MANAGER.demo.account,5;
```

- Opens the database "BASE1.DEMO.ACCOUNT" with the password "MANAGER" in mode 5.
- The domain name may be entered in upper or lower case.

```
>base=histra = histry
Password  =>>
Open mode =>>
```

- Opens "HISTRA", in the current group and account with the local name "HISTRY".
- Prompts for a password and mode.

```
>data-base loanap.loanap;
```

- Opens the database "LOANAP.LOANAP", with no password or mode specified.
- There is no prompt because the command was terminated with a semi-colon.

UNIX Examples:

```
>B membrs, pswd;
```

- Opens the database "MEMBRS" with the password "pswd" in mode 5.

```
>B=123.123.123.123/dbname,WRITEPASS,1;
```

- `Opens the database "dbname" with the password "WRITEPASS" in mode 1 (write/share).`
- The database is opened on a remote system with the specified ip addr.

DEFINE Command [DEF]

This command is used to open (define) a single database or file. This command is an interactive command that prompts for the name of the domain and its associated parameters.

```
DEFINE
```

The DEFINE command is used to open an Image or Eloquence database or file. This command is a sub-set of the OPEN command. It provides backwards compatibility with QUERY and previous versions of Ask. We recommend that the OPEN command be used instead of the DEFINE command.

The operation of this command varies slightly depending on whether or not a domain has already been opened. If no domain is open, the define command will open the specified domain. If at least one domain has been opened, the define command will change the parameters associated with the first domain, by closing and re-opening the domain with the new parameters. If a carriage return is entered, the value of the current parameter is not changed. The define command prompts for the following parameters:

name	is the name of the database or file to be opened.
password	is the password that specifies the level of access to be granted to the current user.
mode	is the mode to be used when opening the domain
data-sets	is a priority list of data sets. See the DATA-SET command for more information.
OUTPUT	specify the current output file. See the OUTPUT command for more information.

See the OPEN command for more information about the name, password and mode parameters.

Examples:

```
* There are no domains open.
>DEFINE
Domain    =>>basea
Password  =>>
Open mode =>>5
Data-Sets =>>m-customer
Output    = TERM
           >>
```

After the command is complete, the domain basea is opened in mode 5 and the list of sets with top priority is initialized to m-customer. The default output device remains as the current standard list (terminal).

```
* Now run the command again with a database open.
>DEF
Data Base =basea           , basea
          >>
Password =>>
Open mode =5
         =>>1
Data-Sets =m-customer
         =>>m-customer,d-order
Output   = TERM
         =>>
>
```

After the command is called for the second time, the first base (basea) is closed and then re-opened in mode 1 (shared access in read and write). The list of priority sets has been changed.

See also: `OPEN`, `DATA-SETS`, `OUTPUT`, `SUPERDEFINE`.

SUPERDEFINE Command [SU]

Similar to the DEFINE command, this command is used to open one or more domains. This command will interactively prompt for domain parameters to open a domain. The SUPERDEFINE command will continue to prompt until all domains have been opened.

```
SUPERDEFINE
```

The SUPERDEFINE command is used to open an Image or Eloquence database or file. This command is a sub-set of the OPEN command. It provides backwards compatibility with previous versions of Ask. We recommend that the OPEN command be used instead of the SUPERDEFINE command.

The operation of this command varies depending on whether or not a domain has already been opened. If no domain is open, the command will open the specified domain(s). If at least one domain has been previously opened, the superdefine command will change the parameters associated with the open domain(s), by closing and re-opening the domain(s) with the new parameters. If a carriage return is entered, the value of the current parameter is not changed. The superdefine command prompts for the following parameters:

name	is the name of the database or file to be opened.
password	is the password that specifies the level of access to be granted to the current user.
mode	is the mode to be used when opening the domain

See the OPEN command for more information about these parameters.

This command will continue to prompt for domains until a carriage return or "//" is entered as the name of a domain.

Special Name Syntax:

<c/r>	If a carriage return is entered, the current value of the name (or other parameter) is maintained. If all open domains have been processed, a <c/r> will stop the command.
//	Same as <c/r>.
/D	Delete (Close) the current domain and all domains which follow. The superdefine command will stop.
/E	Exit (interrupts) the superdefine command with no further changes.

Examples:

```
* No domains are currently open.
>su
Domain    =>>membrs
Password = >>
Open mode =>>
Domain    =>>membr2
Password = >>
Open mode =>>1
Domain    =>>
```

This process continues until the user gives a null response (carriage return or //). In the above example, two databases are opened with the specified parameters.

```
* Now run the command again with 2 databases open.
>su
Data base = membrs, MEMBRS
          >>membrs.report
Password =>>
Open mode = 5
          >>
Data base = membr2, MEMBR2
          >>
Password =>>
Open mode = 1
          >>5
Domain    =>>loanap.loanap
Password = >>
Open mode =>>
Domain    =>>//
```

Makes the changes to the open domains and opens one additional domain.

See also the `DEFINE` command.

2.3 CLOSE Command [CL]

Close a domain.

```
CLOSE { name | ALL }
```

This command closes (removes access to) one of the *domains* opened using a previous OPEN (DATA-BASE, DEFINE or SUPERDEFINE) command. All of the data sets and items belonging to this domain become inaccessible following the close command.

Where:

name	is the short name of the domain to be closed. If a local name has been given when opening the domain, then you must use the local name to close the domain.
ALL	A special form of the close command which closes all of the open domains.

Example:

```
>open basea.data,,mode = mybase;
... various commands ...
>close mybase
```

You can close all domains currently open using the special format:

```
>close all
```

The 'SHOW FILES' command will display the list of currently open domains.

2.4 DATA-SETS Command [S]

Specify a priority list of sets to use when locating an item.

```
{DATA-S[ETS]} [=] [set [,set ...] ]
{S[ETS]     }

(Default: empty list)
```

Where:

set	Is the name of a dataset or file. You may enter a single dataset or a list of sets separated by commas. All of the sets specified are considered "high priority" sets when qualifying items.

Discussion:

Whenever an item (or set) name is used, AskPlus must determine the location of the item (or set) in the available domains. Since the same name can be used for different items in different sets or domains, you must specify the exact location of these items when you refer to them.

You can do this in three ways:

a) By *fully qualifying* the item (in other words, by specifying the name of the set to which it belongs and even the name of the domain),

b) By using the DATA-SETS command which lets you define an ordered list of sets which will be scanned with top priority to locate an item. If the item being located belongs to **only one** of the sets in the list, it will be automatically used; otherwise the list of possible sets (from the list of priority sets) is displayed and you are prompted to make a selection interactively as in c) below. Similarly, if the item cannot be located in this list of sets, AskPlus will prompt for the location as explained below.

c) By specifying the location interactively: this means that if an item does not have a unique location, AskPlus displays the list of sets to which this item belongs and prompts you to make a choice interactively. In batch mode, AskPlus will select the first item in the list which, in most cases, is probably the wrong choice. When executing in batch, it is important to make sure that all items are uniquely identified.

Examples:

```
>open membrs;
```

```
>set=loan-file
>find collateral ib 100, 199

>set=member-file, loan-file
>find account = 123456
ACCOUNT  is a member of these sets (or files):
 1  MEMBRS.MEMBER-FILE
 2  MEMBRS.LOAN-FILE
Which one do you wish to use ?1
```

Notes:

- The set priority list can also be configured using the `DEFINE` command.
- The '`SHOW set'` command displays the current priority list.

2.5 FORM Command [FO]

Display the information available on the structure of any open domains.

```
FORM     [domain            ]
         [[domain.]set      ]
         [[[domain.]set.]item]
         [SETS  [, domain ] ]
         [ITEMS [, domain ] ]
         [PATHS [, domain ] ]
```

The FORM command is used to obtain structural information about one or more of the currently open domains.

Where:

<<none>>	Display all information (sets & items) about all open domains. The information displayed is the same as 'FORM *domain*' for all open domains.
domain	Display all information (sets & items) about the specified domain. The information displayed is the same as 'FORM *set*' for all sets in the domain.
set	Display all information (items) about the specified set. This option lists all the items along with their specifications for each set whose name is the same as the one entered. Also displays the capacity of each set and the number of entries it contains. An asterisk indicates a key item. If a 'I' appears after the asterisk, it specifies an IMSAM sorted key. If a 'O' is present, the item is an OMNIDEX key.
item	Displays information about the specified item in all of the sets where the item is located.
SETS	Displays a list of all of the sets in the specified domain. No item information is included.
ITEMS	Displays a list of all of the items in the specified domain. All items are included, no set information is provided.
PATHS	Displays information about the paths between data sets.

Examples:

```
>fo m-customer
```

```
          Data Base: BASEA.DEMOE.ASKPLUS


          Set Name: M-CUSTOMER,Manual

          Items:
                 * CUST-NO,              I1
                   CUST-NAME,            U20
                   CUST-ADDRESS,         U20
                   CUST-ZIPCODE,         U6
                   CUST-TOT-ORDER,       P12

                Capacity: 40              Entries: 8
```

```
    >fo date


          Data Base: BASEA.DEMOE.ASKPLUS


          Set Name: D-ENTRIES,Detail

          Items:
                   DATE,                 U6

          Set Name: D-ORDER,Detail

          Items:
                   DATE,                 U6
```

```
>fo sets


          Data Base: BASEA.DEMOE.ASKPLUS


                                   Item            Entry Entry
          Sets:              Type Count Capacity Count
          Length


             M-SUPPLIER        M     3     40        5     14
             A-PRODUCT         A     1     40        8      2
             M-CUSTOMER        M     5     40        8     27
             A-ORDER           A     1     40       17      2
             D-ENTRIES         D     4     60       29     15
             D-PRODUCT         D     4     40        8     18
```

```
   D-ORDER          D    5     60      17     10
   D-SUPPLIER       D    3     48      15      5
```

```
>fo items
```

```
Data Base: BASEA.DEMOE.ASKPLUS


Items:

   SUP-NO                 I1
   SUP-NAME               U20
   SUP-ZIPCODE            U6
   PROD-NO                I2
   CUST-NO                I1
   CUST-NAME              U20
   CUST-ADDRESS           U20
   CUST-TOT-ORDER         P12
   CUST-ZIPCODE           U6
   ORDER-NO               I2
   DATE                   X6,           xx/xx/xx
   COMMENT                U16
   AMOUNT                 Z12
   PRICE                  I2
   STOCK-QT              2I2
```

```
>fo paths
```

```
Data Base :BASEA.DEMOE.ASKPLUS

Path Identifying Information

                  Associated
Master Set Name   Detail Set Name   Search Item
Name  Sort Item Name

M-SUPPLIER        D-ENTRIES         SUP-NO
                  D-SUPPLIER        SUP-NO

A-PRODUCT         D-PRODUCT         PROD-NO
                  D-ORDER           PROD-NO
                  D-SUPPLIER        PROD-NO

M-CUSTOMER        D-ORDER           CUST-NO

A-ORDER           D-ORDER           ORDER-NO
```

```
Path Identifying Information

Associated
Detail Set Name   Search Item Name  Sort Item Name
Master Set Name

D-ENTRIES        !SUP-NO
M-SUPPLIER

D-PRODUCT        !PROD-NO
A-PRODUCT

D-ORDER          !ORDER-NO
A-ORDER
                  CUST-NO
M-CUSTOMER
                  PROD-NO
A-PRODUCT

D-SUPPLIER       !SUP-NO
M-SUPPLIER
                  PROD-NO
A-PRODUCT

The "!" indicates the primary path.
```

2.6 Working with Files

AskPlus is able to open, read and report from files as well as from databases. Using files in AskPlus is very similar to using a dataset or table. Some files are self-describing which means that the layout, or structure, of the file is defined within the file. However, many files do not contain this information. For these files, since there are no items (or fields), you must know the layout of the data in the file and you must tell AskPlus how to interpret the data stored in the file.

Opening a File

Files are opened using the same syntax as a database. However, since there is normally no password or mode associated with a file, these parameters can usually be omitted:

```
OPEN filename [=localname];
```

- If specified, the localname will be used as the name of the file for all future AskPlus commands.
- If the ';' is used to terminate the command, AskPlus will not prompt for a password or mode.

Note: On the HP3000 (MPE) file names must be specified using MPE syntax. The POSIX filename syntax is not supported in AskPlus/3000.

SUPRTOOL SD Files

AskPlus recognizes SUPRTOOL SD (LINK) files on both MPE and UNIX systems. On UNIX systems, SUPRTOOL will create two separate files. The first file will have the name specified by the OUT statement, the second will have the same name with a ".sd" extension. When opening the file in AskPlus, you must specify the name of the ".sd" file. For example, to open a file called SUPRLINK created by SUPRTOOL, use the following open command:

```
OPEN SUPRLINK.sd;
```

Specifying Item Names

If a file is in a self-describing format recognized by AskPlus, you can refer to the items in the file in exactly the same way as you would refer to an item in a dataset or table. To see if AskPlus has recognized the file, use the FORM command:

```
FORM FILENAME
```

If AskPlus displays a list of individual item names, then the file has been recognized and you can use standard AskPlus syntax to access the individual items:

```
ITEM_NAME                    -or-
FILENAME.ITEM_NAME
```

If the file is not a self-describing file, or if AskPlus does not recognize the file, you must use sub-item syntax, or you must teach AskPlus the layout of the file.

Using Sub-Item Syntax

One method of extracting a field from a record, is to use the same syntax that is used to extract a portion of a field. Using this syntax, you must specify the offset (in the record), the type and the length. Use this syntax:

```
FILENAME.(offset:typelen)
```

Where:

offset	is the offset of the field in the record (first field starts at offset=1)
typelen	is the type and length of the field. The type must be one of (X, U, I, J, P, K, R, E) and the length is the length according to the specified type.

For example, let's look at a file (MBRDATA) which contains the following 3 fields. The ACCOUNT (I2), the NAME (X30) and the BALANCE (I2). The fields ACCOUNT and BALANCE are I2 which means they occupy 2 16 bit words (4 bytes). To access these fields, you would use the following syntax:

MBRDATA.(1:I2)	This is the ACCOUNT.
MBRDATA.(5:X30)	This is the NAME.
MBRDATA.(35:I2)	This is the BALANCE.

Using an INI File

If you are going to be using the same file frequently, you may want to setup an INI file to describe the layout of the file. The INI file acts as a mini dictionary for the file and the file effectively becomes a self-describing file.

The INI file is structured like a typical Windows INI file with multiple sections. Each section consists of a keyword followed by a value. A ';' is used as a comment.

Each INI file must begin with a Header section:

```
[Header]
WordWay={MPE | Little Endian}
```

```
NBFields=nbr_of_fields
Reclen=rec_len_in_bytes
DataFile=filename
NBRecs=nbr_of_records
DataOffset=0
```

Where:

WordWay	is the byte order for numeric data. Use the value that makes sense for the file that you are reading. If the file was created for MPE systems, use the keyword “MPE”. For UNIX (and other Little Endian systems), use “Little Endian”.
NBFields	is the total number of fields in the file.
Reclen	is the total number of bytes in each record.
DataFile	is the name of the file that actually contains the data. On MPE systems you must use an MPE filename, not a POSIX filename.
NBRecs	is the total number of records in the file. If this value is not specified it is calculated based on the RecLen.
DataOffset	Is the offset where the data starts. Use 0.

After the header section, there is a separate Field section for each different field:

```
[Field#]
Name=field_name
Type=type
Len=len_in_bytes
Occurs=1
Offset=offset_in_bytes
```

Where:

Field#	Is the number of each field. The first section is called Field1, the second Field2 and so on.
Name	is the name of the field.
Type	is the data type. Must be one of (X, U, I, J, P, K, R, E)
Len	Is the length of the field **in bytes**. Note: this is different from sub-item syntax where the length is specified in the type. For example, for an I2 field, use a length of 4 bytes.
Occurs	is the number of times the field occurs. This is usually 1, but can be greater than 1 for compound (array) fields.
Offset	is the offset of the field in the record. The offset is zero-based,

meaning the first field starts at offset=0, not 1. Note: This is also different from sub-item syntax, where the first byte is equal to 1.

The [Field#] section is repeated for each field in the file.

INI File Example:

A sample INI file is shown below for the file that was described in the previous section.

```
[Header]
WordWay=MPE
NBFields=3
Reclen=38
DataFile=MBRDATA
DataOffset=0

[Field1]
Name=ACCOUNT
Type=I
Len=4
Occurs=1
Offset=0

[Field2]
Name=NAME
Type=X
Len=30
Occurs=1
Offset=4

[Field3]
Name=BALANCE
Type=I
Len=4
Occurs=1
Offset=34
```

Opening the INI file:

Once you have created the INI file, you open the INI file in AskPlus. AskPlus recognizes the INI file syntax and then automatically opens the data file specified in the Header section. Use the following syntax:

```
OPEN MBRINI = MBRDATA;
```

Special Note for HP 3000 (MPE) Users:

The INI file feature was originally designed for UNIX systems and therefore it requires that both the INI file and the DATA file be byte stream files. The HP 3000 provides a utility called TOBYTE that converts regular (fixed length) MPE files to byte stream files. This utility must be run once to convert both the INI and DATA files. The TOBYTE utility is accessed from the POSIX shell.

Converting the INI file:

```
:sh
>mv INIFILE INIFILE.tmp
>tobyte –at ./INIFILE.tmp INIFILE
```

This creates a new byte stream file called INIFILE and saves the original file as INIFILE.tmp.

Converting the DATA file:

```
:sh
>mv DATAFILE DATAFILE.tmp
>tobyte ./DATAFILE.tmp DATAFILE
```

This creates a new byte stream file called DATAFILE and saves the original file as DATAFILE.tmp.

3 Selecting Data

3.1 Overview

Data selection is probably the most important step in the creation of a report. During the selection, or query as it is sometimes called, AskPlus selects the data records (rows) to be included in the subsequent report or save commands. Information about the selected data is stored in the select file. AskPlus uses the information stored in the select file whenever it executes a report (or a save) command. For all domains except Oracle, data is selected using the FIND command. For Oracle domains, use the CHOOSE statement in the REPORT and SAVE commands.

To properly understand the process of selecting data, it is helpful to define some terms:

Primary	The primary is the dataset (or file) that is used as the starting point for the selection. In Visimage, this is known as the **Primary Topic**. The primary is required and only one primary can be defined in each selection.
Secondary	The secondary is a dataset (or file) logically related to the primary. In Visimage, these are known as **Secondary Topics**. Secondary sets are not required. Multiple secondaries are allowed. In some cases the access path to the secondary must be defined using a link statement.
Extended Entry	When executing the selection, AskPlus starts by reading the primary either serially or by key. For each record in the primary, AskPlus links to corresponding records in each of the secondaries. The logical entity that is formed by the combination of records from the primary and secondary datasets (or files) is known as an extended entry.
Criteria	The selection criteria is a logical condition that specifies which extended entries (records from the primary and secondary topics) should be selected.
Select File	AskPlus uses the select file to store information about the selected records from both the primary and the secondary datasets. The select file does not contain the actual data, but rather information that allows AskPlus to retrieve the selected data whenever needed. The select file is not used for Oracle domains.

The selection process identifies which datasets and/or files (primary sets and secondary sets) will be used in the report. During the selection extended entries which match the specified criteria will be stored (or saved) in the select file. Most report specifications include the criteria which determines which records should be selected. The challenge is to translate this criteria into terms that AskPlus can understand using the items from the primary and secondary datasets and files.

3.2 Data Selection Examples

Listed below are examples of data selections specified in normal conversation, followed by their corresponding AskPlus selection statements.

Overdue Accounts

Request	Show me all customers whose total overdue balance is more than $1,000 and who have not made a payment in the last thirty days.
Assumptions	The customer record contains four balance due items (BAL-CURR, BAL-30, BAL-60, BAL-90) and the date of the last payment is called LAST-PMT-DATE. Balances are stored with two implied decimal points and dates are in YYMMDD format.

Selection	(BAL-30 + BAL-60 + BAL-90) > 100000 AND %DAYS-OLD(LAST-PMT-DATE) > 30

Comments

- We did not use BAL-CURR in the selection because this amount is not overdue.
- We use the DAYS-OLD macro, which calculates the number of days between a date and today's date.
- To check the balance of $1000, we use the value 100000, which includes two implied decimal points. See *Implied Decimals*, later in this chapter.
- This selection would be more difficult if the customer record did not maintain the balance and payment information. In some cases, we would have to look at all the accounts receivable records and do a multiple pass report. See *Multiple Pass Selection* in the *Visimage Reference Manual*.

Approaching Credit Limit

Request	Show all customers who owe more than 80% of their credit limit.

Assumptions	Same as above, plus there is an item called CREDIT-LIMIT. This item does not have any implied decimal points.
Selection	(BAL-CURR + BAL-30 + BAL-60 + BAL-90) > (CREDIT-LIMIT * 80)
Comments	▪ This time we add BAL-CURR because we want the total amount owed. ▪ To calculate the 80% we multiply by 80 and divide by 100. In this case, we did not divide by 100 because the CREDIT-LIMIT does not contain any implied decimals – but the balances do. Thus, multiplying by 80 without dividing by 100 gives the percentage in the correct scale. For example, $10,000 * 80 is 80,000, which is $8,000 with two implied decimals.

Pre-natal Visits

Request	Find all patients who have made a pre-natal visit sometime in the first 6 months of 1996.
Assumptions	A pre-natal visit is a procedure code between 3100 and 3199 and the field DATE-OF-SERVICE is in CCYYMMDD format.
Selection	PROC-CODE.(1:x2) = “31” AND DATE-OF-SERVICE IB “19960101”, “19960630”
Comments	▪ We used a sub-item (first 2 characters for PROC-CODE). We could have used the IB (includes between) operator or a wildcard.

G/L Expenses

Request	List all of the G/L expenses for departments 300 to 320 and 400 to 420 for a specific month. Do not show any expenses in the range 900 to 999.
Assumptions	The department is stored as GL-DEPT and the expense code is a 3 digit GL-EXPENSE. The user will be prompted for the transaction month in the format YYMM and the item TRANS-DATE is stored as YYMMDD.

Selection	(GL-DEPT IB 300,320 OR GL-DEPT IB 400,420) AND NOT (GL-EXPENSE IB 900,999) AND TRANS-DATE.(1:X4) = ?TRANS-PERIOD-YYMM

Comments

- Notice the OR between GL-DEPT in the range 300 to 320 and 400 and 420. In the verbal request, we asked for departments ranging from 300 to 320 **AND** 400 to 420.
- However, in the AskPlus criteria, we changed it to **OR**. Why? As a data record is evaluated by AskPlus, the **AND** logic would check that the first condition is true AND, at the same time, the second condition is true. This cannot be. A single record cannot have a GL-DEPT in the 300 range AND in the 400 range at the same time. Therefore, we want to select records when either the 300 range condition is true, **OR** when the 400 range condition is true. That way, we will get both groups.
- When using the OR operator, it is very important that you use parentheses to group your criteria. Parentheses are used to control the order of operations in the selection. Statements enclosed in parentheses will be evaluated first.
- For the GL-EXPENSE condition, there was no “not includes between” operator, so we used IB and then prefixed the entire condition with a NOT.

Exceptional Donations

Request	Find all donations from corporate donors that exceeded $10,000 and from individual donors that exceeded $500.
Assumptions	The donor type of C is a corporation and I is an individual. The date of the donation is stored in CCYYMMDD format and the amount contains two implied decimal points.

Selection	(DONOR-TYPE = “C” AND AMOUNT / 100 > 10000) OR (DONOR-TYPE = “I” AND AMOUNT / 100 > 500)

Comments	▪ Again we have to use OR to ensure that we correctly select both groups of donors. ▪ Each amount is divided by 100 so that the scale matches the dollar volume we are looking for.

Missing Immunizations

Request	Find all children who are two years of age and have not received any of their two-year immunizations.
Assumptions	The item DOB is the date of birth. The procedure codes 5137, 5142 and 5168 all indicate the correct immunization. If any of these procedures have been done, we do not want to select this patient.
Selection	%AGE-IN-YEARS(DOB) = 2 AND !RANGE((PROC-CODE <> 5137,5142,5168),”@”)
Comments	▪ The AGE-IN-YEARS macro calculates someone’s age in years ignoring months. ▪ The !RANGE operator allows a selection to be based not just on the current record but on a group of records belonging to a single entity -- in this case, a patient. This syntax finds patients who have no records with these procedure codes. In other words, all (@) of the records are not equal to the specified codes. This operator is described in more detail later in this chapter.

It is possible to "comment out" certain lines of the selection criteria by placing << ... >> brackets around the desired text. At least one line of the criteria must remain active.

3.3 Implied Decimals

We have looked at several selection examples involving amounts containing implied decimals. In many databases, amounts are stored with a number of implied decimals. For example, if a dollar amount is stored with 2 implied decimals, the value $1.23 is actually stored as 123. Since the database contains a whole number (no decimal points) you must modify your selection values accordingly.

You will need to verify which items in your database use implied decimals. Also, remember that implied decimals will affect other calculations. For example an amount with one implied decimal multiplied by an amount with 2 implied decimals, yields a result with 3 implied decimals. To select all extended prices over $500, you would need to specify:

```
QUANTITY * PRICE > 500000
```

Calculations Involving Implied Decimals

Working with implied decimals is fairly easy if you remember the following rules of arithmetic.

- If you multiply values containing implied decimals, the number of implied decimals in the result is the sum of the number of implied decimals in the calculation.
- If you divide values containing implied decimals, the number of implied decimals in the result is the number of decimals in the numerator, minus the number of decimals in the denominator.
- If you want to reduce the number of implied decimals in a calculation, divide by 10 for every decimal that you wish to remove. In other words, to remove one decimal divide by 10, to remove 2 divide by 100, to remove 3 divide by 1000 etc.

3.4 FIND Command [F]

The FIND command is used to select records from one or more *sets*.

```
FIND [SEARCH] statement[;] [statement[;] [...]] [ END]
```

The `FIND` command is composed of the FIND keyword, followed by one or more *statements*. The statements, each one of which is explained in detail in the following sections, specify the exact parameters that AskPlus will use to perform the selection. The following statements (or keywords) may be included in a FIND command.

SEARCH This keyword specifies that comparisons with character values containing *wildcard* characters will be made using the `SMART` option. If SEARCH is not specified, you must use the (SMART) option for each value containing *wildcards*. See ***Wildcard Constants*** (Page 212) in *Chapter 8, Expressions & Formats* for more information.

PRIMARY [=] *set*

Used to specify the name of the primary set (or file). To refine a previous selection, use the INSEL statement instead of PRIMARY. If the PRIMARY set is not explicitly identified, AskPlus will choose the primary based on the rules described below.

INSEL [=] { * | *file* }

Used to specify the name of a select-file to be used as the **input** for another selection. If an INSEL is specified, AskPlus will apply the new selection to the records stored in the specified select file.

OUTSEL [=] [* | *file*] [(@ | *set1* [*,set2* [,...]])]

Used to specify which sets should be included in the **output** select file created by this selection. Also used to specify the name of the select-file. This statement is not required, but is usually needed for selections involving more than one set.

LIMIT [=] nn

Specifies the maximum number of extended entries to select. If the limit is reached, the selection will stop.

SORT S*n*, *expression* [, { ASC | DES }] [, DATE]

Used to specify the sort keys for the selection. While AskPlus allows a sort to be specified during the selection, it is recommended that the sort be done in the REPORT (or SAVE) command which

	follows.
LINK	*name, step* [: *step* [...]] Describes an access path between two sets. Link statements are required if there is no default path (join) between the primary set and the secondary sets.
NEWREG	[#]*name* : [*scale*] *typelen* [[=] *expr*] Used to create and optionally initialize a *register*. Registers, more commonly called variables, can be used in the selection to perform intermediate calculations. This statement is executed only once during the selection to create and initialize the variable.
#variable	= *expression* [,IF *condition*] Describes the calculation to be executed for the *variable.* This statement will be re-evaluated for every *extended entry*, even if the entry does not match the selection criteria. The variable is calculated **before** the selection criteria is evaluated.
criteria	A logical condition which is applied to each *extended entry* in order to determine whether or not it qualifies. If the condition evaluates to TRUE, the extended entry will be selected. The criteria, if specified, must be the last statement in the FIND command. There is no theoretical limit to the number of objects evaluated in a condition nor to its complexity in terms of nesting level (number of parentheses and number of logical or arithmetic operators).
END	This keyword signifies the end of the find command. This statement is optional, the find command will terminate automatically as soon as a complete selection criteria has been specified.

PRIMARY Statement [PRIMARY]

Explicitly specify the name of the primary set. AskPlus always reads the primary set first, before reading any records in the secondary sets.

```
PRIMARY [=] [domain.]set
```

Where:

domain	Is the name of the domain containing the primary set. The domain name is optional and is only required to fully qualify the set name, when the set exists in more than one open domain.
set	Is the name of the primary dataset or file.

Discussion:

- The primary set is the one which will be read first. If possible, the primary set will be read by key. If no key values have been specified, the primary set will be read serially. For each primary entry that qualifies for the selection criteria, AskPlus will read each of the secondary sets. Once it has been determined that the extended record meets or cannot meet the selection criteria, AskPlus will proceed to the next secondary entry. When all possible secondary entries have been read, AskPlus will read the next primary entry.
- The primary set, is also the set with the highest priority when resolving item names. If an item name, which is not fully qualified, is specified and the item exists in the primary set, AskPlus will use the item from that set.
- This statement cannot be used with the INSEL statement. The PRIMARY statement specifies which set (or file) should be read first. The INSEL statement specifies which select file should be used as the starting point for the selection.
- If no OUTSEL statement is specified, only records from the primary set will be stored in the select file.
- If the PRIMARY statement and the INSEL statement are not specified, the primary set will default to the first set mentioned in some other statement of the find command.
- Although rarely used (except in the case of Oracle domains), the PRIMARY statement is also available in the REPORT and SAVE commands.

Examples:

Specify the primary:

```
>F PRIMARY=LOAN-FILE
>>BALANCE > 0
```

- In this example, the primary set is specified as LOAN-FILE.
- Even though the item BALANCE exists in several sets, AskPlus will automatically use LOAN-FILE.BALANCE in the selection criteria.
- Since no OUTSEL was specified, only records from LOAN-FILE will be saved in the select file.

No primary specified:

```
>F LOAN-FILE.BALANCE > 0
```

- This example is functionally equivalent to the previous example. The primary will be LOAN-FILE because this is the first set mentioned in the FIND command.

No primary and no other set specified:

```
>F BALANCE > 0
BALANCE  is a member of these sets (or files):
 1  MEMBRS.SHARE-FILE
 2  MEMBRS.LOAN-FILE
 3  MEMBRS.LOAN-ADDENDUM
Which one do you wish to use ?2
```

- In this example, no primary set is specified. Since BALANCE is not fully qualified, AskPlus prompts for the set to be used. In this example, the set selected was LOAN-FILE. LOAN-FILE is now the primary set.
- Note, however, that in batch (where no prompting is possible), the primary file will default to SHARE-FILE (first set listed) instead of LOAN-FILE. Since this will produce completely different and probably unexpected results, make sure that you fully qualify ambiguous items and sets.
- Since no OUTSEL was specified, only records from LOAN-FILE will be saved in the select file.

Using a priority set instead of the Primary:

```
DATA-SET=LOAN-FILE
>F BALANCE > 0
```

- This example is very similar to the previous example, however, since the DATA-SET command was used to specify LOAN-FILE as a priority dataset and since BALANCE exists in LOAN-FILE, LOAN-FILE will be automatically chosen as the primary set.

Different primary specified:

```
>F PRIMARY=MEMBER-FILE
>>LOAN-FILE.BALANCE > 0
```

- This example is completely different from the previous examples. In this case, the primary is explicitly declared as MEMBER-FILE. This means that AskPlus will serially read the MEMBER-FILE set and then check to see if that member has at least one LOAN-FILE record with a BALANCE greater than zero.
- The select file will store information about qualifying members (MEMBER-FILE), but not their loans.
- The selection will take longer to execute, because AskPlus has to read all of the MEMBER-FILE records **and** all of the associated LOAN-FILE records.

INSEL Statement [INSEL]

Specify the name of an input select-file to be used as the starting point for this selection.

```
INSEL [=] { * | filename }
```

Where:

*****	Use the previous (or current) select file.
filename	Use the select file with the name specified by filename.

Discussion:

- In some respects, the INSEL statement can be thought of as a special case of the PRIMARY statement. Instead of reading a set as the primary, AskPlus starts the selection by reading a previously created select-file. The purpose of the INSEL statement, is to save processing time by refining an existing selection instead of starting from scratch.
- The special character '*' can be used in place of a file name and corresponds to the name of the current (or most recently used) select-file. You can change the current select-file using the `SELECT-FILE` command or the `OUTSEL` statement.
- If the INSEL statement is used, the PRIMARY statement has no meaning and is not allowed.
- The INSEL statement is similar to using the Query SUBSET command.
- If the INSEL statement and the PRIMARY statement are not specified, the primary set will default to the first set mentioned in some other statement of the find command.

Examples:

The following example shows an initial query that selects a group of loan records. These records are saved in a select file called #primsel (the primary selection). After these records are used in a report, two different sub-selects are run that start with the primary selection. Each of these sub-selects create a new select file called #subsel. The overall performance of this will be faster than running three separate queries.

In this example, the names of the select files are stored in the variables #primsel and #subsel. By using variables and the MAKETEMP command, AskPlus will generate unique filenames to use for the select files each time the report is run.

```
>* Create 2 registers to hold the names of
>* the temporary select file.
>newreg primsel:x100
>maketemp #primsel
>newreg subsel:x100
>maketemp #subsel

>* Execute a selection using loan-file
>find primary=loan-file;outsel=#primsel(@)
>>control-flags(11) = 1 and balance > 0 and
>>rest-flag-11 <> 80

<< run a report using all of the selected records >>

* Now refine the selection looking for specific
collateral codes
>find insel=#primsel;outsel=#subsel
>>collateral ib 101, 199 or collateral ib 301, 399

<< run another report using these collateral codes >>

* Now refine the original selection again looking for
different codes
>find insel=#primsel;outsel=#subsel
>>collateral ib 201, 299 or collateral ib 401, 499

<< run another report using these collateral codes >>
```

OUTSEL Statement [OUTSEL]

Specify the name of the **output** select-file generated by this `FIND`, and the list of sets which are stored in the select-file.

```
OUTSEL [=] [ filename ] [( set-list | @ )]
```

Where:

filename	The first parameter is the name of the select-file created by the current find. The filename can be an actual filename, or it can be the name of an AskPlus global variable (register) which contains a file name. To load a variable with a unique (temporary) filename, use the maketemp command. If no filename is specified, AskPlus will use a default (unique) file name.
(set-list)	The second parameter is the list of sets that will be stored in the select-file. The list of sets (or files) must be separated by commas and enclosed in parentheses. If links are being used, the link name must prefix the set name. See the examples below.
@	The '@' is a special form of the set-list. This symbol means that 'all' sets mentioned in the find command should be included in the select-file. This is the syntax most commonly generated by Visimage and QTOASK.

Discussion:

The `FIND` command selects entries according to the selection criteria, and then stores a reference to the qualifying (extended) entries in the select-file. Subsequent commands (REPORT, SAVE, FIND) then read the select-file to process the data. Only sets mentioned in the OUTSEL statement's set-list are actually stored in the select-file. Therefore, it is very important that your OUTSEL statement includes all of the sets that you will need.

The OUTSEL statement is used to specify a permanent filename for the select-file. This is useful, if you plan to use the INSEL statement in a future selection. More importantly, the OUTSEL statement can be used to specify the list of sets which are to be stored in the select file. By default, AskPlus stores only the pointers for the *primary* set which is determined automatically or by using the `PRIMARY` or `INSEL` statement. Using OUTSEL, you can provide a list (in parentheses) of the sets that will be stored in the select-file. The special character '@' can be used in place of the list of sets to specify that **all** the of the sets mentioned anywhere in the find command should be saved.

Each set name must use the following syntax rules (*dataset*):

```
[linkname..][database.]set
```

If a set is accessed through a *link*, you must specify the name of this link along with the name of the set. If the set exists in more than one open domain, fully qualify the set name with the database name.

The following examples, show how changing the OUTSEL will effect the find and subsequent commands.

Examples:

No Select File specified:

```
>f primary=member-file
>> loan-file.balance > 0
```

- Since no OUTSEL is specified, the select-file will only store information from the primary set, in this case MEMBER-FILE.
- This selection finds members with at least one loan with a balance greater than zero, but the select-file does not contain any references to the qualifying loans.
- If a report is written that includes fields from loan-file, AskPlus will display all of the related loan records, even if the balance is equal to (or less than) zero.

Adding OUTSEL=(@):

```
>f primary=member-file;outsel=(@)
>>loan-file.balance > 0
```

- This OUTSEL statement specifies that all sets (MEMBER-FILE and LOAN-FILE) are to be stored in the select-file.
- This selection finds members with at least one loan with a balance greater than zero, it also finds all of the qualifying loans.
- If a report is written that includes fields from loan-file, AskPlus will display only loan records that have a balance greater than zero.
- This form of the OUTSEL statement is the most common.

Using OUTSEL prior to Deleting records:

```
>FIND SEARCH PRIMARY = ATM-FILE
>>OUTSEL = (ATM-FILE);
>>(ATM-FILE.MEMBER-STATUS <> 1 AND
>> LOAN-FILE.COLLATERAL = 111 )
```

```
>DELETE
```

- In this example, the use of OUTSEL is VERY important. The DELETE command will delete all of the records from all of the sets that are stored in the select-file. Since we want to delete only ATM records and not LOAN records, we must be careful to limit the OUTSEL to only one set.

Specifying OUTSEL with a link:

```
>FIND SEARCH PRIMARY = PLEDGE-FILE
>>OUTSEL = (L1..LOAN-FILE);
>>LINK L1, PLEDGE-FILE.LOAN-ACCOUNT :: LOAN-
FILE.ACCOUNT
>>(criteria)
```

- This example shows the use of a link. Since the OUTSEL contains a specific set list (and not the '@'), the name of the link (in this case L1) must be specified.

LIMIT Statement [LIMIT]

Specifies the maximum number of extended entries that can be selected in the FIND or printed in the REPORT or SAVE. When the limit is reached, the command will stop.

```
LIMIT = nn
```

Where:

nn	Is the maximum number of records to select (FIND) or read (REPORT/SAVE) commands.

Discussion:

- The limit must be a positive (non-zero) number.
- The limit must be specified before the selection criteria.

Examples:

Setting a Limit

```
>FIND LIMIT=10
>>MEMBER-FILE
```

- This command will select the first 10 entries of the members file, which will be read serially.

- or -

```
>FIND PRIMARY=LOAN-FILE
>>LIMIT=100
>>END
```

- In this case the limit is 100 records. The limit may be specified anywhere as long as it is **before** the selection criteria.
- In this example, the END keyword is used because there is no selection criteria.

SORT Statement [SORT]

Specify the sort sequence to use when making a selection. It is strongly recommended that you **do not** use the sort statement when selecting data. Since AskPlus does not know how many records will be selected, it is difficult to correctly estimate the size of the sort file needed to sort the data. Use the SORT statement in the REPORT and SAVE commands instead.

```
SORT S[n],expr [, { ASC | DES } ] [, DATE]
```

Where:

n	Sort break level (or sort order number). The sorts with the **highest** number are evaluated with the highest priority; the other sorts are sub-sorts within the major sort. If *n* is omitted, the sort order is the order in which the sort statements are specified. Un-numbered sorts always have a lower priority than numbered sorts.
expr	The value of *expr* will be used as the field to be sorted. The expression will be evaluated for each *extended entry* meeting the selection criteria. In many cases, *expr* is the name of an item. The expression may involve items and registers (variables).
ASC	The sort is sorted in *ascending* order. In ascending order, numbers are sorted from smallest to largest and letters are sorted from a-z and from A-Z. This is the default and does not need to be specified.
DES	The sort is sorted in *descending* order. In descending order, numbers are sorted from largest to smallest and letters are sorted from Z-A and from z-a.
DATE	Indicates that the key value is a date whose format depends on its assignment in `DATEFORMAT` (see `ASSIGN`).

Example:

```
>FIND primary=loan-file
>>sort s2, account
>>sort s1, suffix, DES
>>collateral ib 100,199
```

LINK Statement [LINK]

Describe the access path AskPlus will use to link the primary dataset to a secondary set.

```
LINK linkname, step [ : step [ ... ] ]
```

A link is used to **explicitly** describe an access path (join) from the primary to one or more secondary sets or files.

The link definition consists of a link name, followed by one or more link **steps**. Each link step describes the link between the current set (or file) and the next set (or file). Each step has the following definition:

```
expr :: [[domain.]set.]key [link-parm]
```

The optional link-parm has the following syntax:

```
{FORWARDS | BACKWARDS} [COUNT = nn]
```

Where:

linkname	Is the name of the link followed by a comma. The linkname is used when referring to the name of an item. The syntax is: `linkname..[set.]item`
expr	Is any item or expression which can be evaluated from the currently accessible fields. For a field to be accessible, it must either belong to the primary, belong to a secondary set accessible via a previously defined or default link or belong to a secondary set whose access path has already been defined in the current link.
domain	Is the new domain accessed by this step. If the set name is unique, the domain name is optional.
set	Is the new set accessed by this step. If the key name is unique, the set name is optional. Use of the set name is recommended for clarity.
key	Is the name of a key field that provides keyed access to the set. The set will be accessed for all the values of the *key* equal to the value of *expr*. If the linked set is organized in *indexed sequential* form, the *key* may be a *partial key*, in other words, composed of the first characters of a full key. On MPE systems, if the linked set is a **flat** file with no access key, the expression is evaluated as a record number and the file is read directly according to this number.
direction	The keyword FORWARDS indicates that the chain should be read

from first to last. This is the way that AskPlus normally reads chains. The keyword BACKWARDS indicates that the chain should be read in reverse order from last to first. These options are most often used with sorted chains and can be specified on each step of the link. A sorted chain is not required.

COUNT=n	The COUNT keyword causes the chained read to be limited to n records. This parameter can be specified on each step of the link. When specifying a COUNT, you must also specify the link direction: FORWARDS or BACKWARDS. To read the first record in a chain, use "FORWARDS COUNT=1" ; to read the last record only, use "BACKWARDS COUNT = 1".

Discussion:

- Links may be simple one step links from the primary set to one secondary set, or they may consist of multiple steps starting with the primary and joining several different sets together. Each step uses the same basic syntax. Each step is separated from the previous step by a colon.
- In theory, there is no limit to the number of steps which can make up a link.
- When accessing secondary sets in the FIND command, AskPlus performs the equivalent of an outer join. Primary entries are **not** excluded from the selection if no matching secondary entry can be found. To exclude primary entries in this situation, use the special system variable $MISSING as part of your selection criteria.
- The most frequently used links may be defined in the **manager file** which is read automatically if AskPlus is started with either the "-a" or "-u" run line parameters. See the *Visimage Administrator Manual* for more information.
- Links defined in the FIND, REPORT or SAVE commands are considered local to the current command. Global links are created using the ASSIGN LINK ... command.
- A local link will always take priority over a default or global link, provided the link name is specified.
- During a FIND command, if the set that has been accessed via a LINK **is saved in the select file (OUTSEL)**, the LINK will **not** be required in the subsequent REPORT or SAVE. In fact, repeating the link in a REPORT or SAVE could cause invalid results. This is because sets saved in the select file are considered primary sets by all subsequent commands that use the select file.
- It is possible to access the same set more than once via different link

commands. If multiple links are used to the **same** set, this set may be included in the OUTSEL statement only **once**.

Examples

Example #1

```
>find primary = m-customer
>>link products, cust-no :: d-order.cust-no :
>>               prod-no :: d-product.prod-no
>> cust-zipcode = "94230" and
>> products..description = "RACQUET"
```

In this example, the link "products" is used to describe an explicit path between the m-customer set and the d-product set through the d-order set. This link is required, because AskPlus does not know how to access the d-product set from the m-customer set. The link works as follows:

- get the customer number in the primary file;
- for each order placed by this customer (access to the order set by customer number), get the number of the product ordered
- use the product number to access the product set.
- The field (products..description) is then the item (description) of the set (d-product) for the values of product numbers ordered by those customer whose zip code is 94230.

This link is composed of two *steps*. The first step links the primary set (this is always the case) to the d-order set, and the second step goes from the d-order set to the d-product set. For both steps, the *expr* are simply the name of an item (cust-no or prod-no) in the current set.

Example #2

```
>find primary = tagfile; outsel=(@)
>>link mbr, tagfile.(5x9) :: member-file.account
>>link loans, tagfile.(5:x9) :: loan-file.account
>>mbr..member-file.account <> $MISSING and
>>loans..loan-file.suffix = tagfile.(14:x2)
```

In this example, the primary is a tag file that contains the account number in positions 5 through 13 (max 9 digits) and a suffix in positions 14 & 15. Two separate links are defined which link the account number to the member-file (mbr) and to the loan-file (loans).

In the selection criteria, we select all records in the tag file that have a corresponding member in the member-file (<> $MISSING) and we select the loan-file record that has a matching suffix. If there is no matching suffix, the record will not be selected.

Example #3

```
>find primary = MEMBER-FILE;
>>outsel=(addr..ADDRESS-INFO)
>>link addr, #addr-key :: address-info.search-key
>>newreg addr-key:x12
>>#addr-key = "S" cat !format("999999999", ACCOUNT)
cat "00"
>>CONTROL-FLAGS(21) = 1
```

In this example, the link is defined using a register (variable). Notice that the link name must appear in the OUTSEL statement, if you reference the set by name.

Example #4

Example using the Direction and Count parms:

```
>find primary=MEMBER-FILE;OUTSEL=(@)
>>LINK L1, ACCOUNT :: SHARE-FILE.ACCOUNT
           FORWARDS COUNT=1
>>L1..BALANCE < 2500
```

This example, selects all members with less than $25 in their primary (SUFFIX=0) share account. Since the ACCOUNT path between MEMBER-FILE and SHARE-FILE is sorted by the SUFFIX, this link will guarantee that only SUFFIX=0 records are selected. This selection is more efficient than specifying "AND SUFFIX = 0" as part of the selection, but the results will be the same.

NEWREG Statement [NEWREG]

Used to create a **local** variable (register). There is no difference between a register and a variable. In AskPlus, for historical reasons, variables are sometimes called registers. In Visimage and in this manual, for clarity, they are called variables.

```
NEWREG [#]name:[scale]typelen [[=] expr]
```

Where:

name	is the name of the variable. The name must follow the syntax rules for an identifier. It must start with a letter, is limited to 20 characters and may contain, letters, numbers and a few special characters. The # prefix is used to signify that the name that follows is a variable name. This prefix is not required in the newreg statement.
scale	is the number of occurrences of the variable. The default scale is 1 and does not need to be specified. A scale larger than one indicates that the variable is an array.
typelen	Is the type and length of the variable. See Appendix B for a list of valid types and lengths.
expr	Is the initial value of the variable. If no initial value is specified, numeric variables will be initialized to zero and character variables will be initialized to blank. If the variable is an array (has a scale greater than 1), the initial value will be applied to all elements of the array. The special operator "!index(@)" is allowed when loading the initial value of array variables.

Discussion:

- The `NEWREG` statement is used to create a new (local) variable by assigning it a name, scale, type, length and possibly an initial value. AskPlus does allow variables to be used **without** declaring them in a NEWREG statement, however, it is strongly recommended that all variables be declared with a NEWREG statement prior to their use.
- Variables created using this statement are considered **local variables**. A local variable exists only for the current command (FIND, REPORT, SAVE). The value of a local variable cannot be passed from one command to another. Global variables, created with the ***NEWREG command*** *(Page 181),* keep their value until they are purged using the ***PURGEREG command*** (Page 183).

- This statement is available in the FIND, REPORT and SAVE commands (for local variables) and as a **command** when creating global variables.

Examples:

```
>* Example using newreg in the find command.
>find primary=share-history
>>newreg cymd:31i2=(%first-day-pri-mo + !index(@) - 1)
>>entry-date=#cymd(@) and ...
```

The previous example creates an array of 31 values and loads each element of the array with a date from the previous month. The first element is the first of the month, the second element is the second and so on. The find command then searches for ENTRY-DATE equal to one of the specified values. It always loads 31 values even for months that have less than 31 days. (For months with less than 31 days, the FIND will look for invalid dates, but no error will occur and no data will be selected.) This FIND will return the same data as "ENTRY-DATE IB %FIRST-DAY-PRI-MO, %LAST-DAY-PRI-MO", however, there is one major difference. If ENTRY-DATE is a key field, the example using the array will perform a keyed read. The example using IB will always use a serial read. The performance difference is substantial when reading large sets.

Variable Compute Statement [#]

Specify how the value of a variable is to be computed. In the FIND command, each variable statement will be evaluated for every extended entry, regardless of whether or not the entry is selected.

```
#variable[(index)] = expr [,IF cond]
```

Where:

variable	is the name of the variable. The name of the variable **must** be prefixed by the # symbol. This symbol identifies this statement as a compute statement. It is recommended that the variable has already been declared by a previous newreg statement or command. If the variable has not been previously defined, the type and length may be specified as follows: "#variable:[scale]typelen = ...". (See the NEWREG statement for more information on scale and typelen.)
index	is the index to use for array variables. The default index is 1 and does not need to be specified. The index can be a number, an expression, or the special symbol @. If '@' is used, the expression will be evaluated for all elements of the variable.
expr	is the expression to be computed for the variable. The expression can involve items and variables, including the variable that is being evaluated.
cond	is an IF condition to be evaluated prior to the variable expression being evaluated. If the condition is TRUE, the variable will be computed according to the expression. If the condition is FALSE, the variable value is not changed.

Discussion:

- The variable statement may be used to update both local and global variables.
- Provided the IF condition is TRUE, the variable statement will be evaluated for every extended entry that is read by the FIND command, and not just for each qualified entry.
- All variable statements will be evaluated **before** AskPlus evaluates the selection criteria. This means that the variable may be used in the selection criteria and it's value will reflect the current entry being evaluated.
- Similarly, if a computation is conditional (uses the IF option), the condition is evaluated based on items in the current entry, whether this entry will be

selected or not.

Examples:

Example #1 - Calculating Amount over Credit Limit

```
>find primary=loan-file
>>newreg over-limit:i2
>>#over-limit = CREDIT-LIM - BALANCE
>>(#over-limit > 0)
```

Since the first character of the selection criteria is a '#', the condition must be enclosed in parentheses. Without the parentheses, AskPlus would interpret the statement as a compute statement and not as the selection criteria.

Example #2 - Linking a Tag File to the SHARE-FILE

```
> find primary=tagfile;outsel=(@)
>>newreg acct:i2
>>newreg sfx:i1
>>link l1, #acct :: share-file.account
>>#acct = tagfile.(5:x9)
>>#sfx = tagfile.(14:x2)
>>(#sfx = l1..share-file.suffix)
```

Example #3 - Calculating a Tax Amount

```
>find primary=order-lines
>>newreg tax:i1
>>newreg tot_tax_amt:i2
>>#tax = 70
>>#tax = 75,if region = 1
>>#tax = 80,if region = 2
>>#tax = 85,if region = 3
>>#tot_tax_amt = price * quantity * #tax / 100
```

Selection Criteria

The selection criteria is always the last statement of the FIND command. Unlike other statements, the selection criteria does not have a special keyword to begin the statement. The selection criteria is a logical expression that returns the value TRUE or FALSE. It can be a simple expression like (BALANCE > 0) or it can be a complex expression involving many comparisons, ANDs, ORs and parentheses. As soon as the selection criteria is logically complete, the FIND command will be executed.

```
expression
```

Where:

expression	Is a logical expression that determines which extended entries will be selected during the FIND. If the expression evaluates to TRUE the extended entry will be selected. If FALSE, the record will not be selected.

Discussion:

- The expression must be a logical expression that returns TRUE or FALSE.
- The expression may contain ANDs and ORs and any number of parentheses.
- The selection criteria is not required. If no criteria is specified, all of the records from the primary set will be selected.

Examples:

- `MEMBER-FILE.REST-FLAG-11 <> 80, 81`
- MEMBER-FILE.REST-FLAG-11 <> 80, 81 AND
  SHARE-FILE.SUFFIX = 0 AND SHARE-FILE.BALANCE > 2500
- MEMBER-FILE.REST-FLAG-11 <> 80, 81 AND
  (SHARE-FILE.SUFFIX = 0 AND SHARE-FILE.BALANCE > 2500 OR
  LOAN-FILE.COLLATERAL <> 999 AND LOAN-FILE.BALANCE > 0)

-
  ```
  MEMBER-FILE.REST-FLAG-11 <> 80, 81 AND
  SHARE-FILE
  ```

This last example, shows a special case where the name of a set may be used as part of the selection criteria. When used in this way, the set name always evaluates to TRUE. This technique is useful for forcing AskPlus to read a secondary set and include it in the select file.

3.5 FIND ALL Command [F ALL]

This special form of the find command selects all of the entries in a single set.

```
FIND [ALL] [domain.]set
```

The FIND ALL command is a special version of the find command. No records are actually read during the selection. The name of the set is stored in the select-file along with a flag indicating that all records have been selected. The data will be read later when it is actually required by another command.

Where:

ALL	Special keyword instructing AskPlus to select all entries in a set without actually reading the set. If this keyword is omitted, AskPlus will read the entire set and select all of the entries.
domain	The name of the domain. The domain name is only needed to fully qualify the set name.
set	The name of the dataset or file being read.

Discussion:

- No other statements are valid when using 'FIND ALL'.
- The format 'FIND set' (no ALL) is a variant of the 'FIND ALL ...' format; however, in this case the set **is read** sequentially and all the records found are marked in the select file.
- If the "FIND ALL filename" command is used with a file on UNIX, AskPlus will display "-1" records qualified because AskPlus does not know how many records are actually in the file. Use "FIND filename" instead.

Examples:

```
>F ALL ORDERS
Current OUTPUT select file is xxxx
entries qualified 396
```

- No data is actually read.
- The select-file is marked so that all records of ORDERS will be used by the next command that uses the select-file.

```
>F ORDERS
Current OUTPUT select file is xxxx
```

```
using serial read
entries read 396 396
Entries qualified 396
```

- In this example, since the ALL keyword was not used, AskPlus serially reads the ORDERS dataset.

3.6 FX Command [FX]

This command is used to read a dataset that has been indexed using OMNIDEX keys. (HP 3000 Only)

```
FX [base.]set
```

After entering the FX command, the user is prompted to enter a key value for each of the OMNIDEX keys. Criteria must be entered in the syntax described in the OMNIDEX documentation.

An implicit `'AND'` applies between the criteria of each key. When all keys have been entered, the command loops and you can enter new criteria to refine or correct your selection.

When the selection is completed, entering '//' ends the command and creates the select-file. To cancel the command, enter '///'.

```
>FX D-PRODUCT

DESCRIPTION                 =>>SAIL@
50 entries selected

PRICE                       =>>169900:200000
10 entries selected


DESCRIPTION                 =>>
PRICE                       =>>*,180000:190000

2 entries selected


DESCRIPTION                 =>>// << done >>
select file ASKSEL0 created
2 entries qualified

>
```

At this point, you can enter the Report command, just like after any Find command. You can also complete your selection through a regular FIND command using the INSEL=* option.

3.7 The !RANGE Operator

The RANGE operator is a special operator that works on multiple records rather than just the current record. In a normal selection, AskPlus decides whether or not the current record qualifies based on the values of items found in that record. The RANGE operator can be used to base a selection on a group of records. When this operator is applied to a secondary dataset, AskPlus looks at all records in the chain before deciding whether or not to qualify the associated primary record.

Use RANGE to select records from the **primary** dataset as a function of the number of times the specified condition is satisfied in a **secondary** dataset.

The syntax of the RANGE parameter is as follows:

```
!RANGE( (condition), “parm”)
```

Where:

condition	Is the condition that will be evaluation for all of the records in the secondary dataset. The range condition **must** be enclosed in a set of parentheses.

The parm must be of the following form:

“@”	Condition must apply to **every** record.
“n”	Condition must be matched **exactly** the “n” times you specify.
“min/”	The condition must qualify **at least** “min” times.
“/max”	The condition must qualify **at most** “max” times.
“min/max”	The condition must apply **at least** “min” times and **at most** “max” times.
Never	There is no syntax to specify that the condition never applies. In order to select based on a condition never being true, negate the condition and use the “@” parameter. In other words, the condition is not satisfied all the time.

Select File

The rules for creating the select file are slightly different when using the !RANGE operator. Under normal circumstances, the select file will contain all of the chosen topics. This is not possible when using RANGE. When the !RANGE operator is used, AskPlus must look at a **series of records** before deciding whether or not the primary record qualifies. Consequently, AskPlus only saves **primary topic records** in the select file. You should specify "@" as the OUTSEL (Output Select File Definition) when using the !RANGE operator.

In many cases, the primary records are enough for your report requirements. If you need the secondary records as well, perform a second selection using the **Refine Current Selection (INSEL=*)** option. In the second selection, remove the RANGE operator and select based on the range condition itself. See below for an example.

Examples

Find all customers that placed at least 5 orders during a three month period.

Topic	CUSTOMER-MASTER
Condition	!RANGE ((ORDER-DATE IB 940101,940331), “5/”)

This selects and saves customer records in the select file. To select the actual orders, perform a second selection using the Refine Current Selection Option.

Topic	CUSTOMER-MASTER, ORDERS
Condition	ORDER-DATE IB 940101,940331

This second selection will only contain customers that placed at least 5 orders in the specified range (because we are refining a selection). It will also contain all orders placed by those customers in the specified date range.

Find all customers who have never ordered product ABC.

Topic	CUSTOMER-MASTER
Condition	!RANGE ((PART-NBR <> “ABC”), “@”)

By looking for PART-NBR not equal ABC **all** the time. We actually find the customers who have never ordered that particular part.

Note: Remember that the select file only contains primary records. Also, remember to change the Select File Definition to @, or the primary file name.

Select File

The rules for creating the select file are slightly different when using the !RANGE operator. Under normal circumstances, the select file will contain all datasets specified in the OUTSEL statement. If the "@" parameter is used, all datasets used in the FIND will be included in the select file. This is not possible when using RANGE; The dataset included in the RANGE condition cannot be included in the select file. When the !RANGE operator is used, AskPlus must look at a **series of records** before deciding whether or not the primary record qualifies. Consequently, AskPlus only saves **records that are not part of the RANGE condition** in the select file. You may still specify "@" as the OUTSEL, but in this case "@" means all datasets except those included in the RANGE..

In many cases, the primary records are enough for your report requirements. If you need the secondary records as well, perform a second selection using the **(INSEL=*)** option. In the second selection, remove the RANGE operator and select based on the range condition itself. See below for an example.

Examples

Find all customers that placed at least 5 orders during a specified date range.

```
FIND PRIMARY=CUSTOMER-MASTER;OUTSEL=(@)
!RANGE((ORDER-DATE IB ?BEG-DT,?END-DT), “5/”)
```

This selects qualifying customers and saves CUSTOMER-MASTER records in the select file. To select the actual orders, perform a second selection using INSEL=*.

```
FIND INSEL=*;OUTSEL=(@)
(ORDER-DATE IB ?BEG-DT,?END-DT)
```

Since we are using INSEL=*, we no longer need to check how many times the condition is met. We know that all of the customers that we selected have at least 5 orders in the specified date range. This time (since RANGE is not being used) the select file will contain records from both CUSTOMER-MASTER and ORDERS.

Find all customers who have never ordered product ABC.

```
FIND PRIMARY=CUSTOMER-MASTER
!RANGE((PART-NBR <> “ABC”), “@”)
```

In this example, we want to find customers that have never ordered a particular part. This is accomplished by using the "not equals" operator and verifying that the condition is TRUE all (@) the time. There is no need to refine this selection since we have found the customer records that we need.

Find members that have at least one charged off share account (DESC-ABRV="COS") or at least one charged off loan (COLLATERAL=999):

```
FIND PRIMARY=MEMBER-FILE;OUTSEL=(@)
!RANGE((SHARE-FILE.DESC-ABRV = “COS”), “1/”) OR
!RANGE((LOAN-FILE.COLLATERAL = 999), "1/")
```

This example shows different RANGE conditions on two different sets. If either condition is met, the MEMBER-FILE record will be selected.

When using batch mode to execute your selection, it may be necessary to enclose the entire RANGE in a set of parentheses so that the first character of the line is not an "!". Having a "!" as the first character may cause problems on MPE systems or in UC4 scripts, where the "!" is interpreted as a comment!

3.8 Improving Selection Performance

Before discussing how to improve performance, we first have to understand what happens when AskPlus selects data.

- Every selection starts by reading the primary topic. The primary is either a dataset, a file, or the results of a previous selection.
- Whenever possible, the primary should be read by key; otherwise, it will be read serially. A keyed read is usually faster because the records that match the key values can be returned quickly. By contrast, a serial read must check every record in the dataset (usually much slower for large datasets). Only certain fields in the database are defined as key fields, so a keyed read is not always possible.
- All fields from secondary datasets or files will, by design, be read by key.

Example:

Primary:	MEMBER-FILE
Selection:	%YRS-OLD(BIRTH-DATE) < 18 and LOAN-FILE.ENTRY-AMOUNT > 100000

- In this example, the primary (MEMBER-FILE) will be read serially. A keyed read is not possible. The only key fields in MEMBER-FILE are ACCOUNT, SSN and HOUSEHOLD.
- For every member under 18 the LOAN-FILE is read by key (ACCOUNT) and the original loan amount (ENTRY-AMOUNT) will be checked for every loan belonging to this member.

1. **Choosing the primary:**
   Often, choosing the right primary will have the biggest impact on your selection performance. There are several considerations when choosing the primary.

   - Are all secondary sets (needed for this report) accessible from the primary? Secondary sets must be accessible via a keyed read. If a field is a key in one set and not in the other, choose the set that allows a key to be used for the link.
   - Are all possible records available in the primary? ($MISSING)
     A report that links MEMBER-FILE (primary) to LOAN-FILE can include all MEMBERS, even if they do not have any loans. If LOAN-FILE is the primary, members without loans cannot be included in the report.

- Does the selection criteria allow a keyed read?
  If the selection includes a key item (and value) which is connected to the rest of the criteria by "AND", choose the set containing the key item as your primary.
- Choose the smallest set which contains the most restrictive selection conditions.
  When doing a serial read, it helps to select the smallest set. (The set with the fewest records.) However, it is more important to select a set that has a restrictive selection criteria. Choosing a small set will not help, if all of the records in the set meet the selection criteria.
- Is there a TAG file or SD file you can use as the primary?
  If you already have a file that contains the basic information, like a member account number, use this as the primary. You may also want to consider using SUPRTOOL to create an SD (LINK) file to use as your primary. On large datasets, the initial serial read may be faster using SUPRTOOL.

2. **Whenever possible, use a keyed read:**
For large sets (like history) a keyed read will help performance, particularly if the number of entries that qualify for the keyed read is less than 30% of the total records. In SH-HISTORY and LN-HISTORY, the ENTRY-DATE is a key field. If we are able to specify the individual dates that we are interested in using, the selection will be much faster (by key) than if we have to read through all of the history transactions looking for a small number that we are interested in. (See below)

Example:

- Read SH-HISTORY looking for all (command = 100) records since the beginning of the year by members with control-flag(75) on. (Assume that a fairly small number of members have control-flag(75) on.)

Primary:	MEMBER-FILE
Selection:	MEMBER-FILE.CONTROL-FLAG(75) = 1 AND SH-HISTORY.COMMAND = 100 AND SH-HISTORY.ENTRY-DATE >= "20050101"

  - Choosing MEMBER-FILE as the primary makes the most sense in this case, because only a small number of members will be selected. The larger set (SH-HISTORY) is only read for qualifying members, which will be faster than reading the entire history set.

- Read SH-HISTORY looking for all (command = 100) records in the previous month by members with control-flag(75) on. (Again, assume that a fairly small number of members have control-flag(75) on.)

Primary:	SH-HISTORY
Selection:	ENTRY-DATE = #prev-mth(@) AND MEMBER-FILE.CONTROL-FLAG(75) = 1 AND SH-HISTORY.COMMAND = 100
Variable:	#prev-mth (Type: 31I2)
Initial Value:	%load-mth-dates(200505)

- This time, we can do a keyed read on the date, so choosing SH-HISTORY as the primary now makes sense. The syntax (item = array(@)) will do a keyed read for all (@) of the elements of the array, which in this case is all of the days in the month May 2005.

4 Reporting Data

4.1 Overview

One of the primary purposes of AskPlus is creating a formatted report. The REPORT command (discussed in this chapter) is used to create a formatted (ASCII) report that can be displayed, printed or written to a file. The REPORT command always generates ASCII output even when the data is written to a file. See ***Chapter 5, Creating Data Files*** (Page 135) for information on creating binary output files.

A report is created with data extracted from a previously opened domain. See ***Chapter 2, Accessing Data*** (Page 19) for information on opening and closing domains. Except when using Oracle, the data to be included in the report should already have been selected by a previous `FIND` command. See *Chapter 3, Data Selection* for more information on selecting data. Use the CHOOSE statement (see below) to select data for a report using data from an Oracle database.

Command Structure

Like the FIND command, a REPORT command consists of a series of statements terminated by the 'END' statement. Taken together, these statements determine what fields will be included in the report, how the report will be formatted and where the report will be printed/saved. These statements fall into the following classifications.

Output and Control Statements

These statements are used to specify certain global options and to control where the output of the report will be displayed, printed or saved.

Data Selection Statements

These statements determine the source of the data that will be used with the current report. Unless you are working with Oracle, these statements are not normally required. By default, AskPlus will use the data selected during the most recent FIND command as the data source.

Variable and Compute Statements

These statements allow you to create variables and specify calculations using the variables.

Print and Formatting Statements

These statements control exactly what data will be printed and how it will be sorted and formatted.

Report ALL

A special form of the REPORT command, called 'REPORT ALL', provides a quick list of all of the data fields referenced in the current select file. When this form of the command is used, no other statements are valid.

4.2 REPORT Command [R]

The REPORT command initiates the start of a new report. Each report starts with the 'REPORT' keyword and is followed by a series of report statements terminated by the 'END' statement. These statements determine how each report will be executed. There are basically four different types of statements: output & control statements, data selection statements, variable & compute statements and print & format statements. Each of the report statements are described in detail below. All reports must be terminated by the 'END' statement. The data for the report will come from the most recent selection, unless one of the data selection statements is used to specify a different data source. Oracle reports must include at least one data selection statement.

```
REPORT
statement [;]
[ ... ]
END
```

The statements which are part of this command form a powerful *report generation language*. Each statement or group of statements is covered in the following sections. The available statements are as follows:

Output and Control

These statements are used to specify the report output (display, printer or file) as well as global settings like lines per page, default calculation mode, default date format and more.

OUTPUT	Specifies the output destination (display, printer or file)
Control	Control statements specify a large number of global options.
INCLUDE	Specifies the name of a command file to be included.

Data Selection

These statements are used to specify the data source to be used for the report. These statements are necessary for Oracle databases, but are not normally used with other databases and files. By default, the current select file will be used as the data source for the report. The data can be limited with the LIMIT statement and further refined using the CHOOSE statement.

PRIMARY	Specifies the primary dataset to be used.

INSEL	Specifies the name of the input select file.
CHOOSE	Refines the current selection.
LIMIT	Limits the number of records used in the report.
LINK	Specifies an access path to other datasets or indexed files.

Variable/Compute

These statements are used to create temporary variables (registers) that will be used in the current report and also to specify the calculations to be performed with the variables.

NEWREG	Creates a new variable (register).
#variable	Specifies a compute statement.
IF	Calculates the value of an ASK Version C flag. This statement has been replaced by the more general compute statement.

Print/Formatting

These statements specify how the report will be sorted and what data fields will be printed.

SORT	Specifies the sort order.
Print	Specifies how each field should be printed.
EDIT	Specifies an edit mask.
END	This keyword specifies the end of the statements in the `REPORT` command. Except in the special case of the 'REPORT ALL' command, the END statement is always required. It marks the end of the syntax analysis phase of statements and means that the report logic will be checked and the actual output generated.

In the remainder of this chapter, each of these statements will be discussed in more detail in the order that they appear above.

4.3 OUTPUT [OUT]

This command/statement determines where AskPlus output will be sent. If entered as a command (prior to a REPORT), the OUTPUT will be re-directed as specified until the next OUTPUT command. If entered as a statement in the REPORT or SAVE commands, the output will be redirected only for the current command and then will revert back to display (TERM).

```
OUTPUT [=]     { TERM                         }
               { filename [, file_parms]   }
               { LP [, lp_parms]           }
               { $devname  [, lp_parms] }
               { #variable [, parms]          }
(Default: TERM)
```

Where:

TERM	By default, output is displayed on the user's terminal. On the HP3000, the TERM designation is associated with the symbolic file name ASKOUT and is the default equivalent of $STDLIST.
filename	AskPlus output is redirected to a file with the specified name.
LP	By default, represents the system's printer. On the HP3000 this is associated with the symbolic file name ASKLIST and is the default equivalent of the LP class.
$*devname*	On MPE, this option directs output to the specified device name or class. The setting $LP is equivalent to LP. On UNIX, this option redirects output to the printer with the specified devname.
#*variable*	Like the previous two options, this choice redirects output to a file or a device, depending on the value of the variable. AskPlus uses the current value of the variable to determine the output destination. The value of the variable can be a file name, or a $devname.

File Parameters:

File parameters are entered after the filename and separated by a comma. Multiple options may be specified, each one separated by a comma.

DISC=*nnn*	On HP3000 systems, this option determines the maximum number of records (*nnn*) in the output file. On UNIX, this parameter is ignored.
REC=*ll*	This option specifies the maximum length (*ll*) in characters of each record. See also the WIDTH statement in the REPORT Command.
FILL	On UNIX systems, this option causes each output record to be padded (filled) with blanks. The record will be padded up to the specified width or record length. MPE files are always fixed length with padded blanks. UNIX files are variable length and, by default, are not padded with blanks.
APPEND	If the file already exists, new records are appended to the end of the file.
DELETE	If the file already exists, this option will purge the old file and create a new file without being prompted for confirmation.

LP Parameters:

LP parameters are entered after the device name (or LP) and separated by a comma. Multiple options may be specified, each one separated by a comma.

message	On MPE, the specified message will be sent to the master console (operator message) in order to prompt the operator to install a special type of paper (printed forms or labels, for example).
COPIES=*nn*	Specifies the number of copies to be printed of the same report. Default value is 1.
OUTPRI=*nn*	Specifies listing output priority (1 to 13). The default value is 8. MPE only.

Discussion:

- The `OUTPUT` **command** re-directs output from the REPORT and FORM commands as well as from some utility commands.
- All error messages, warnings and prompts are sent to `ASKOUT` and are not redirected by the `OUTPUT` command.
- An output assignment remains in effect until the current command is terminated, or in the case of the OUTPUT **command**, a new OUTPUT **command** is entered.

Examples:

```
>out = lp
>form
>out = term
```

```
>out $lp1
>report
  ...
>>end
```

Example on MPE using a file equation:

```
:FILE ASKLIST=FILE1;DEV=DISC;LIMIT=100000
:ASKPLUS PUBTCP
>out = lp
...
```

Example on UNIX using system variables:

```
export ASKOUT=FILE1
/ASKPLUS/askplus
>newreg #askout:x50=^ASKOUT
>out = #askout, delete
...
```

4.4 Control Statements

Report control statements are used to set global options that are valid for the current report.

```
keyword [[=] parameters]
```

The following statements are available in the REPORT command to define report execution parameters.

PAUSE	The report output is paused after each page awaiting an <Enter> on the terminal. This option is used to display a report page by page on the screen or to load a sheet of paper between each page on a printer, for example if special forms are being used with manual feed.
LINES [=] *nn*	Defines the logical length (nn) of each page. When the maximum number of lines is reached on the current page a new page is begun. The default value is 60 for a report being sent to a file or printer and 23 for the terminal. The minimum number of lines per page is 10. Use LINES=0 to create a file with one logical page of infinite size.
NOPAGE	On MPE, this option deletes the form feed character when starting a new page. This corresponds to the `NOCCTL` option when defining the ASKLIST output file (:`FILE ASKLIST` = ...;`NOCCTL`). Each page will be filled with blank lines rather than using a page eject control character. This option may be useful with certain printers but is especially designed for generating files when these files are to be used by other programs or applications.
WIDTH [=] *nn*	Defines the logical width of the page (number of columns) used for printout. The default value is 132 for a printer and 79 for a screen. Any output line longer than the configured width will be truncated when printed. This option also lets you define the length of records when printing data to a file. See also the rec= parameter of the OUTPUT statement.
WINDOW [=] *nn*	Defines the first logical column of the printout. This is used to display the right-hand part of a report on a peripheral using a width less than the actual report width. For example, you can print out a report 250 columns wide with non compressed characters by requesting two successive printouts: the first one

	with options WIDTH=125;WINDOW=1 and the second with options WIDTH=125;WINDOW=126, then place the two listing side by side.
PAGES [=] *nn*	On MPE, this option is used to force a new spoolfile every *nn* pages. If you have a very large report to print, this option avoids creating large *spool* files and lets you print out the whole report in several batches. This option is ignored on UNIX.
EMPTY [=] "*text*"	Specifies the text or value which will be printed in place of **missing** items when the value of an expression is $MISSING. By default, $MISSING items are treated as blank (for character fields) or zero (for numeric fields).
DYNAMICTRAILER	If this option is specified, the trailer (Z) lines will be printed immediately after the last detail or total (or even group) lines on the page and not at the bottom of the page.
NOSPLITxxx	A global option that tells AskPlus not to split a group of lines of the same type by a page break. In other words, to check **before** printing whether there is enough room on the page to print the group of lines (plus the trailer ('Z' lines)) and otherwise to go to the next page before printing the lines. There are three different keywords: NOSPLITDETAIL, NOSPLITGROUP and NOSPLITTOTAL.
- DETAIL	Means go to the next page if all the 'D' lines for a **single** entry cannot be printed on the current page.
- GROUP	Same thing for group level lines. Refers to all of the GROUP lines of a single sort break.
- TOTAL	Same thing for total lines. Refers to all of the TOTAL lines of a single sort break.
COMMERCIAL	When using this option, AskPlus default numeric variables will be created using an numeric (P20) type. This option cannot be used with the SCIENTIFIC option.
SCIENTIFIC	When using this option, AskPlus default numeric variables will be created using a floating point numeric type. This option cannot be used with the COMMERCIAL option.
ROUND	Causes AskPlus to treat the "/" division operator like the DIVR and return a rounded result. By default, the result of a division

	using "/" will be truncated and "DIVR" will be rounded. This option is not recommended, since many calculations assume that the result will be truncated. Use DIVR instead if you wish to force a rounded result.
COMPRESS	If this option is specified, the report will be generated in compressed mode on all line printers that accept the following escape sequence : '*Esc*&k2S*Esc*&l8D'
USA	When printing the current date, the USA (MDY) format will be used. This option has the same effect as the global command `'ASSIGN DATEFORMAT = USA'` but is temporary to the current command.
EUROPE	When printing the current date, the European (DMY) format will be used. Same as `'ASSIGN DATEFORMAT = EUROPE'` but is temporary to the current command.
INTERNATIONAL	When printing the current date, the International (YMD) format will be used. Same as `'ASSIGN DATEFORMAT = INTERNATIONAL'` but is temporary to the current command.
TAB*n* [=] *cc*	Sets the value of a tab stop. The column number *(cc)* is assigned to the specified tab number (*n*). Tabs can be used in place of numerical print positions in print statements. In most cases the column positions for tabs are calculated automatically by AskPlus based on the preceding tab in order to leave enough space for the largest item printed relative to this tab, plus one character. No actual tab characters are used, AskPlus simply calculates an appropriate position for each column printed using Tabn.

4.5 INCLUDE Statement [I]

Includes (and executes) statements from a file of pre-defined instructions.

```
INCLUDE filename
```

Where:

filename	Is the name of a file containing a list of statements to be included in the current command.

Discussion:

If a sequence of AskPlus statements is to be used by several different report commands, it may be more practical to create a separate file containing these statements and include this file each time it is needed.

An include file is similar to an XEQ file, but has one significant difference. An XEQ file can only be executed as a command. An include file is different because the include file may be specified in the middle of a FIND, REPORT or SAVE command. The include file may contain any statements that are valid for the current command. Include files can be nested. This means that a command file can include other files. When AskPlus reaches the end of a file, control returns to the previous file or to the user's terminal if it is the first level.

Example :

Example printing a standard header:

```
>report
>>newreg title1:x40 = "Title of my report"
>>newreg title2:x40 = "Sub Title"
>>include std-header
>>...
>>end
```

In this example, the std-header file contains print statements to print a standard report header, with a title, page number, date and time as follows:

```
h1, "Date:" join $DATE8, 1 left
h1, #title1, 40 center
h1, "Page:" join $pageno, 79 right
h2, "Time:" join $TIME
h2, #title2, 40 center
h3, %cu-name, 40 center, space a1
```

This *statement* is valid in the FIND, REPORT and SAVE commands. The included file must contain statements that are valid in the context where the file is used.

4.6 PRIMARY Statement [PRIMARY]

Specifies the name of the primary set to be used in the current REPORT or SAVE command. This statement will over-ride the current select file and is not normally used, except with Oracle databases.

```
PRIMARY [=] {[domain.]set|table}
```

Where:

domain	When specified, is the name of the domain containing the primary set.
set	Is the name of the primary dataset or file. The set name may be fully qualified.
table	Is the name of the Oracle table which will be read first.

Discussion:

- Except when using Oracle, the PRIMARY statement is not normally used. By default, the primary is determined by the current select file and no primary statement is required.
- If a primary statement is entered, the primary set will be read first. Unless a CHOOSE or LIMIT statement is entered, AskPlus will report all data from the specified primary.
- This statement cannot be used with the INSEL statement. The PRIMARY statement specifies which set (or file) should be read first. The INSEL statement specifies which select file should be used.

Examples:

```
>REPORT
>>PRIMARY=LOAN-FILE
>>CHOOSE, (BALANCE > 0)
```

- In this example, the primary set is specified as LOAN-FILE.
- The CHOOSE statement will restrict the report to just those loans with a balance greater than zero.
- When reporting from databases and files (other than Oracle), the FIND command should generally be used.

4.7 INSEL Statement [INSEL]

Specify the name of an input select-file to be used as the starting point for this report.

```
INSEL [=] { * | filename }
```

Where:

*****	Use the previous (or current) select file.
filename	Use the select file with the name specified by filename.

Discussion:

- The special character '*' can be used in place of a file name and corresponds to the name of the current (or most recently used) select-file. You can change the current select-file using the `SELECT-FILE` command or the `OUTSEL` statement.
- If the INSEL statement is used, the PRIMARY statement has no meaning and is not allowed.
- If the INSEL statement and the PRIMARY statement are not specified, the report will use the previous (or current) select file. If there is no select file, the primary set will default to the first set mentioned in some other statement of the command.
- In most cases, no INSEL or PRIMARY statement is necessary, since the report will use the current select file. Specifying "INSEL=*" has the same effect as not using an INSEL (or PRIMARY) statement.

Example:

```
FIND PRIMARY=MEMBER-FILE
CONTROL-FLAGS(72) = 1 AND
REST-FLAG-11 NE 80, 81

FIND INSEL=*;OUTSEL=(@)
LOAN-FILE.BALANCE > 0 AND
LOAN-FILE.REST-FLAG-11 NE 80, 81
```

4.8 CHOOSE Statement [CH]

This statement performs a (temporary) selection within the current `REPORT` or `SAVE` command.

```
CH [,] criteria
```

Where:

criteria	is a logical expression that specifies which records are to be included in the current command.

Discussion:

- If a previous selection has been done using the FIND command, the CHOOSE statement temporarily refines the current select file, by applying the criteria to the currently selected records.
- Only those records that match the criteria will be used in the current REPORT or SAVE.
- The select file is **not updated** by the CHOOSE command.
- If no prior selection has been made, the CHOOSE statement completely replaces the FIND command, however, no select file is created.
- The CHOOSE statement **must be used** when specifying a selection criteria for an **Oracle** database. The CHOOSE statement is equivalent to the WHERE clause of an Oracle SELECT statement.

Examples:

```
>FIND LIMIT=100;SHARE-FILE

>REPORT
>>CH (SUFFIX = 0)
>>TF1, SUFFIX, tab1, count
>>END

using previous select file

      54

>REPORT
>>CH (SUFFIX <> 0)
>>TF1, SUFFIX, tab1, count
>>END
```

```
using previous select file
      46
```

These examples show how the choose can be used to change the currently selected records without modifying the contents of the select file. In the FIND, 100 records are selected from SHARE-FILE. In the first REPORT, the CHOOSE restricts the selection to (SUFFIX = 0). When the count is displayed, 54 records have been selected. In the second report, the CHOOSE specifies (SUFFIX <> 0). In this case, the count is 46 records.

4.9 LIMIT Statement [LIMIT]

Specifies the maximum number of extended entries that can be selected in the FIND or printed in the REPORT or SAVE. When the limit is reached, the command will stop.

```
LIMIT = nn
```

Where:

nn	Is the maximum number of records to select (FIND) or read (REPORT/SAVE) commands.

Discussion:

- The limit must be a positive (non-zero) number.
- The limit must be specified before the selection criteria.

Examples:

Using a Limit in a Report

```
>report
>>limit=20
>>d1, balance, tab1
>>...
>>end
```

4.10 LINK Statement [LINK]

Describe the access path AskPlus will use to link the primary dataset to a secondary set.

```
LINK linkname, step [ : step [ ... ] ]
```

A link is used to **explicitly** describe an access path (join) from the primary to one or more secondary sets or files.

The link definition consists of a link name, followed by one or more link **steps**. Each link step describes the link between the current set (or file) and the next set (or file). Each step has the following definition:

```
expr :: [[domain.]set.]key [link-parm]
```

The optional link-parm has the following syntax:

```
{FORWARDS | BACKWARDS} [COUNT = nn]
```

Where:

linkname	Is the name of the link followed by a comma. The linkname is used when referring to the name of an item. The syntax is: `linkname..[set.]item`
expr	Is any item or expression which can be evaluated from the currently accessible fields. For a field to be accessible, it must either belong to the primary, belong to a secondary set accessible via a previously defined or default link or belong to a secondary set whose access path has already been defined in the current link.
domain	Is the new domain accessed by this step. If the set name is unique, the domain name is optional.
set	Is the new set accessed by this step. If the key name is unique, the set name is optional. Use of the set name is recommended for clarity.
key	Is the name of a key field that provides keyed access to the set. The set will be accessed for all the values of the *key* equal to the value of *expr*. If the linked set is organized in *indexed sequential* form, the *key* may be a *partial key*, in other words, composed of the first characters of a full key. On MPE systems, if the linked set is a **flat** file with no access key, the expression is evaluated as a record number and the file is read directly according to this number.
direction	The keyword FORWARDS indicates that the chain should be read

from first to last. This is the way that AskPlus normally reads chains. The keyword BACKWARDS indicates that the chain should be read in reverse order from last to first. These options are most often used with sorted chains and can be specified on each step of the link. A sorted chain is not required.

COUNT=n The COUNT keyword causes the chained read to be limited to n records. This parameter can be specified on each step of the link. When specifying a COUNT, you must also specify the link direction: FORWARDS or BACKWARDS. To read the first record in a chain, use "FORWARDS COUNT=1" ; to read the last record only, use "BACKWARDS COUNT = 1".

Discussion:

- Links may be simple one step links from the primary set to one secondary set, or they may consist of multiple steps starting with the primary and joining several different sets together. Each step uses the same basic syntax. Each step is separated from the previous step by a colon.
- In theory, there is no limit to the number of steps which can make up a link.
- When accessing secondary sets in the FIND command, AskPlus performs the equivalent of an outer join. Primary entries are **not** excluded from the selection if no matching secondary entry can be found. To exclude primary entries in this situation, use the special system variable $MISSING as part of your selection criteria.
- The most frequently used links may be defined in the **manager file** which is read automatically if AskPlus is started with either the "-a" or "-u" run line parameters. See the *Visimage Administrator Manual* for more information.
- Links defined in the FIND, REPORT or SAVE commands are considered local to the current command. Global links are created using the ASSIGN LINK ... command.
- A local link will always take priority over a default or global link, provided the link name is specified.
- During a FIND command, if the set that has been accessed via a LINK **is saved in the select file (OUTSEL)**, the LINK will **not** be required in the subsequent REPORT or SAVE. In fact, repeating the link in a REPORT or SAVE could cause invalid results. This is because sets saved in the select file are considered primary sets by all subsequent commands that use the select file.
- It is possible to access the same set more than once via different link

commands. If multiple links are used to the **same** set, this set may be included in the OUTSEL statement only **once**.

Examples

The LINK statement is most commonly used in the FIND command. If the linked set has already been included in the select file (during the FIND), then the link **should not be repeated** in the subsequent REPORT or SAVE. There are, however, circumstances, when you may wish to use a LINK in the REPORT or SAVE commands. The most common reason, is when you want to have multiple links to the same set. Here are some examples:

Example #1 - Student Database

In this example, imagine a student database that contains course IDs in a compound (array) item called COURSES. We want to display, the course ID and it's description which is found in a separate dataset called COURSE-INFO. The report will display the descriptions for the first 3 courses:

```
>report
>>s1, student-id
>>link L1, courses(1) :: course-info.course-id
>>link L2, courses(2) :: course-info.course-id
>>link L3, courses(3) :: course-info.course-id
>>...
>>g1, student-id, 1 left
>>d1, courses(1), 18
>>d1, L1..course-description, 20 left
>>d2, courses(2), 18, if (courses(2) <> " ")
>>d2, L2..course-description, 20 left, if (courses(2)
<> " ")
>>d3, courses(3), 18, if (courses(3) <> " ")
>>d3, L3..course-description, 20 left, if (courses(3)
<> " ")
>>...
>>t1, " ", 1 left
>>end
```

By using 3 separate link statements, we are able to print the course description for 3 different courses. This link must be done in the REPORT (or SAVE), not the FIND, because we are using multiple links to the same dataset.

Example #2 - Summit Database

In this example, we want to print the name of both the original and current branch for a member's share file accounts. We do this using two link statements.

```
>find primary = share-file
>>...
```

```
>report
>>link orig, orig-branch :: branch-file.branch
>>link curr, curr-branch :: branch-file.branch
>>d1, account, "99999999", tab1
>>d1, suffix, "99", tab2
>>d1, balance, "$$$,$$$,$$9.99-", tab3
>>d1, orig..branch-file.name, tab4
>>d1, curr..branch-file.name, tab5
>>...
>>end
```

Example #3 - AMISYS Database

In this example, we want to print name information for both the subscriber and the provider. Once again, it makes sense to use a link in the REPORT command and not in the FIND.

```
>find primary = services
>>...
```

```
>report
>>link provider, "P " cat prov# :: address.addr-who
>>link subscriber, "S " cat member# :: address.addr-
who
>>...
>>d1, prov#, tab1
>>d1, provider..address.name, tab2
>>d1, member#, tab3
>>d1, subscriber..address.name, tab4
>>...
>>end
```

4.11 NEWREG Statement [NEWREG]

Used to create a **local** variable (register). There is no difference between a register and a variable. In AskPlus, for historical reasons, variables are sometimes called registers. In Visimage and in this manual, for clarity, they are called variables.

```
NEWREG [#]name:[scale]typelen [[=] expr]
```

Where:

name	is the name of the variable. The name must follow the syntax rules for an identifier. It must start with a letter, is limited to 20 characters and may contain, letters, numbers and a few special characters. The # prefix is used to signify that the name that follows is a variable name. This prefix is not required in the newreg statement.
scale	is the number of occurrences of the variable. The default scale is 1 and does not need to be specified. A scale larger than one indicates that the variable is an array.
typelen	Is the type and length of the variable. See Appendix B for a list of valid types and lengths.
expr	Is the initial value of the variable. If no initial value is specified, numeric variables will be initialized to zero and character variables will be initialized to blank. If the variable is an array (has a scale greater than 1), the initial value will be applied to all elements of the array. The special operator "!index(@)" is allowed when loading the initial value of array variables.

Discussion:

- The `NEWREG` statement is used to create a new (local) variable by assigning it a name, scale, type, length and possibly an initial value. AskPlus does allow variables to be used **without** declaring them in a NEWREG statement, however, it is strongly recommended that all variables be declared with a NEWREG statement prior to their use.
- Variables created using this statement are considered **local variables**. A local variable exists only for the current command (FIND, REPORT, SAVE). The value of a local variable cannot be passed from one command to another. Global variables, created with the ***NEWREG command*** *(Page 181)*, keep their value until they are purged using the ***PURGEREG command*** (Page 183).

- This statement is available in the FIND, REPORT and SAVE commands (for local variables) and as a **command** when creating global variables.

Examples:

```
>* Example using newreg in the report command.
>find primary=loan-file;outsel=(@)
>>collateral ib 100, 799 and balance > 0

>report

>>newreg coll-range:x8

>>newreg coll-bal:p20

>>newreg brnch-bal:p20

>>newreg tot-bal:p20

>>...
```

See the Compute Statement (following pages) for the complete report example.

4.12 Compute Statement [#variable]

Describe a computation statement for a variable (register).

```
#variable[(idx)] { = | op } expr [,options] [,level]


(Default: level D computation)
```

Where:

#variable	is of the name of the variable that will be affected by this compute statement. It is strongly recommended that this variable has been previously created by a NEWREG statement. If this is a new variable, it is possible to specify the type, length and scale using a special syntax described below. Otherwise, AskPlus will create a variable using a default type and length.
idx	For array variables, an index may be specified which specifies which element of the array should be affected by the compute statement. The index may be a constant, a variable, an expression or the special index '@'. When '@' is used, the compute statement will be repeated for ALL elements of the array.
{= | *op*}	When equals is specified, the variable will be loaded with the value of the expression. This is the most common form of the compute statement. For compatibility with older versions of ASK and with QUERY, an operator may also be specified. See below for more information.
expr	Is the expression to use with the function. In most cases (when equals is specified), the value of the expression will be loaded as the current value of the variable.
options	A variety of options are allowed which effect how the compute statement will be executed. Options can be used to control when (or if) the compute statement will be executed. The options are described below.
level	Specifies at which level the compute statement should be executed. See below for more information. See also the SORT statement.

Variable Name Syntax:

It is recommended that a NEWREG statement be used to create a variable before it is used in a compute statement. However, AskPlus does allow the scale, type and length of the variable to be included in a compute statement, provided that it is included only once, the first time that the variable is used in an expression.

```
#variable:[scale]typelen ...
```

Where:

scale	When greater than 1, is the number of occurrences of an array variable.
typelen	Is the type and length of the variable. See the NEWREG statement for more information on scale and typelen.

If the variable is not explicitly defined, a default definition is used by AskPlus. This default type/length depends on the first operation in which the variable is used. When the operation gives a boolean result, the variable is created as boolean. When the statement involves character strings, the variable is created as "character" and when the operation is an arithmetic expression, the variable is created either in P20 (packed decimal) or in R4 (real double precision) depending on the context (COMMERCIAL or SCIENTIFIC).

Operator Syntax:

The most common operator is the equals '='. When equals is specified, the current value of the variable is set equal to (loaded with) the current value of the expression. The following syntax is also valid:

```
,{ LOAD | ADD | SUB | MPY | DIV | UPDATE } [ALL],
```

Where:

LOAD	Assigns (loads) the value of the *expression* to the register. Same as: `#var = expr`
ADD	Adds the value of the *expression* to the current value of the register. Same as: `#var = #var + (expr)`
SUB	Subtracts the value of the *expression* from the current value of the register. Same as: `#var = #var - (expr)`

MPY	Multiplies the value of the *expression* by the current value of the register. Same as: `#var = #var * (expr)`
DIV	Divides the current value of the register by the value of the *expression*. Same as: `#var = #var / (expr)`
UPDATE	This operator is used to update the value of a field with the current value of the variable. When using UPDATE, the expression must be simplified to a database item or sub-item. See ***Chapter 6, Updating Data*** (Page 153) for more information.
ALL	If the `ALL` option is used with one of these *operators* the calculation is **immediately** executed for all entries in the *secondary* chain. Otherwise only the first entry in the chain is used. This is different from the `ALL` and `MERGE` keywords discussed below because the calculation is done before executing any of the print statements.

Each of the operators (LOAD, ADD, SUB, MPY, DIV and UPDATE) can be abbreviated to a single character.

Options:

The following options are also valid with the compute statement.

```
[,IF cond] [,level] [, {MERGE | ALL}]
```

cond	The compute statement is only evaluated if the *condition* is `TRUE`. The condition will be evaluated before the compute statement is evaluated. When the condition is TRUE, the compute statement is executed.
level	Specifies at what *level* the compute statement should be executed. The available options are : D, G*nn*, T*nn and* TF (Where nn refers to a previously defined sort break.) The default level is D (detail) which means the compute statement will be executed for every selected record.
ALL	Execute the compute statement for all the entries in the *secondary* chain. This option is only meaningful if at least one field in the expression is from a secondary dataset. The execution is performed in the same loop as the `ALL` print

	statements. Variables calculated with `ALL` should be printed with `ALL`.
MERGE	Same as the `ALL` option, except that the execution is performed in the `MERGE` print loop and not the `ALL` loop. Variables calculated with `MERGE` should be printed with `MERGE`.

Discussion:

- A compute statement may involve local and/or global variables. At the end of the command, local variables are deleted, however, global variables continue to exist and retain the same value that they had when the command terminated.
- In most cases, compute statements are executed for each selected entry. It is however possible to specify that the computation be done on a sort break (either at the beginning of a group (G*nn*), at the end of a group (T*nn*), or at the end of the report (TF).
- The computation may depend on the value of another logical type expression (*condition*) which may involve the value of another variable previously evaluated and/or which may be executed during a *sort break* at level *n*. If a condition `AND` a break level are specified, the computation will only be done if both cases are met (condition true `AND` break).

Order of Execution:

All variable computations are done **before** the variables are used in the print statements, irrespective of the relative position of the computation and print lines in the command. However, computation statements themselves are executed **in the order** in which they appear (for a given sort level).

A special form of the print statement (see below) may be used to reset variables to zero during the execution of print statements.

Examples:

```
>>#full-name = lastnm SPLICE "," JOIN firstnm
>>#tax = 1055,if prod-type = 1
>>#tax = 1186,if prod-type = 2
>>#tax = 1333,if prod-type = 3
>>#amount_per_1000 = quantity * price * #tax
```

Examples:

```
>* Example using newreg in the report command.
>find primary=loan-file;outsel=(@)
>>collateral ib 100, 799 and balance > 0

>report

>>newreg coll-range:x8

>>newreg coll-bal:p20

>>newreg brnch-bal:p20

>>newreg tot-bal:p20

>>#coll-range=" "

>>#coll-range="100-199", if collateral ib 100, 199

>>#coll-range="200-299", if collateral ib 200, 299

>>#coll-range="300-399", if collateral ib 300, 399

>>#coll-range="400-499", if collateral ib 400, 499

>>#coll-range="500-599", if collateral ib 500, 599

>>#coll-range="600-699", if collateral ib 600, 699

>>#coll-range="700-799", if collateral ib 700, 799

>>#brnch-bal = 0, g2

>>#coll-bal = 0, g1

>>#coll-bal = #coll-bal + BALANCE

>>#brnch-bal = #brnch-bal + #coll-bal, t1

>>#tot-bal = #tot-bal + #brnch-bal, t2

>>s2, curr-branch

>>s1, #coll-range

>>g2/1, "Branch:" join curr-branch, 1 left

>>t1/1, #coll-range, 3 left

>>t1/1, #coll-bal, 30, e1

>>t2/1, "Branch Total:", 3 left, space a1

>>t2/1, #brnch-bal, 30, e1
```

```
>>tf1, "CU Total", 1 left
>>tf1, #tot-bal, 30, e1
>>e1, "$$$,$$$,$$9.99-"
>>end
```

This example calculates a collateral code range and then calculates total balances for each range, each branch and a total for the CU.

4.13 IF Statement [IF]

The IF statement is used to set and clear flags based on the value of a condition. A flag is a logical variable that has the value of TRUE of FALSE. This statement provides compatibility with ASK Version C and has been replaced by the more general compute statement discussed in the previous section.

IF, *condition* THEN { SET | CLEAR} *flag* [ELSE {SET | CLEAR } *flag*]

Where:

condition	The condition is an expression that returns the value of TRUE or FALSE. If the result is TRUE, the THEN clause is executed. If the result is FALSE, the ELSE clause, if present, is executed
SET	Sets the value of the flag (or logical variable) that follows to TRUE.
CLEAR	Sets the value of the flag (or logical variable) that follows to FALSE.
flag	A flag is a logical variable created by the NEWREG statement, or it is one of the 100 pre-defined flags named F0 through F99.

Discussion:

- This statement tests the value of a *condition* and sets or clears a *flag* according to the result.
- This statement is supported by AskPlus for compatibly with Ask Version C. AskPlus allows the creation of logical registers which can be calculated using the register compute statement and consequently the IF statement is not necessary.
- The IF statement may be used with Ask Version C flags F*nn* or with any logical register created via the NEWREG statement. A logical register has the type *K1*.
- The flag that is set (or cleared) by the ELSE clause may be the same **or different** from the flag set (or cleared) in the THEN clause.
- This statement is actually a special register computation statement. You can alternate register computation statements with IF statements.

Examples:

Using the IF statement:

```
>>newreg negamt:k1
>>newreg tot-neg:i2
>>newreg tot-pos:i2
>>if, amount < 0 then set #negamt else clear
#negamt
>>#tot-neg = #tot-neg + amount, if negamt
>>#tot-pos = #tot-pos + amount, if not negamt
>>d1, amount, 50, if negamt
>>d1, amount, 70, if not negamt
>>tf1,#tot-neg, 50
>>tf1,#tot-pos, 70
```

Using a Compute statement instead of the IF:

```
>>newreg negamt:k1
>>newreg tot-neg:i2
>>newreg tot-pos:i2
>>#negamt = (amount < 0)
>>#tot-neg = #tot-neg + amount, if negamt
>>#tot-pos = #tot-pos + amount, if not negamt
>>d1, amount, 50, if negamt
>>d1, amount, 70, if not negamt
>>tf1,#tot-neg, 50
>>tf1,#tot-pos, 70
```

Using IF logic without any flags:

```
>>newreg tot-neg:i2
>>newreg tot-pos:i2
>>#negamt = (amount < 0)
>>#tot-neg = #tot-neg + amount, if (amount < 0)
>>#tot-pos = #tot-pos + amount, if (amount >= 0)
>>d1, amount, 50, if (amount < 0)
>>d1, amount, 70, if (amount >= 0)
>>tf1,#tot-neg, 50
>>tf1,#tot-pos, 70
```

4.14 SORT Statement [S]

Specify the sorts (expressions) to use in the report.

```
S[n], expr [, { ASC | DES } ] [, DATE]
```

Where:

n	Sort break level (or sort order number). The sorts with the **highest** number are evaluated with the highest priority; the other sorts are sub-sorts within the major sort. If *n* is omitted, the sort order is the order in which the sort statements are specified. Un-numbered sorts always have a lower priority than numbered sorts.
expr	The value of *expr* will be used as the sort. The expression will be evaluated for each *extended entry* saved in the select file. In many cases, *expr* is the name of an item. The expression may involve items and registers (variables).
ASC	The sort is sorted in *ascending* order. In ascending order, numbers are sorted from smallest to largest and letters are sorted from a-z and from A-Z. This is the default and does not need to be specified.
DES	The sort is sorted in *descending* order. In descending order, numbers are sorted from largest to smallest and letters are sorted from Z-A and from z-a.
DATE	Indicates that the sort is a date whose format depends on its assignment in `DATEFORMAT` (see `ASSIGN`).

Discussion:

- This statement is used to specify the values of the sort keys for sorting data.
- Each sort generally includes a level or sort order number (S3, S2, S1). The highest level number is considered the major sort. Each lower sort level will be sorted within the higher level sorts. If no level is specified, the sorts are evaluated in the order they appear.
- The sort levels are used by the **print** and **compute** statements to determine when these statements should be executed.

Example:

```
>FIND PRIMARY=LOAN-FILE
>>COLLATERAL IB 100, 199
```

```
>REPORT
>>s2, account
>>s1, balance, DES
>>h1, "Loan Info: C-Code between 100 & 199", 1 left
>>h2, "Sorted in descending order by balance.",
      1 left, space a1
>>h3, "Sfx  Coll       Balance", 20 left, space a1
>>g2/1, "Account:" join account, 1 left
>>g1/1, suffix, "99", 20 left
>>g1/1, collateral, "999", 25 left
>>g1/1, balance, "$$$,$$$,$$9.99-", 30 left
>>t2/1, balance, "$$$,$$$,$$9.99-", 30 left, add
>>end
```

4.15 Print Statements

Print statements are used to print each element of the report. Each print statement specifies the item or expression to print along with information on where the item is to be printed and how the item should be printed.

Each print statement has the form:

```
level, expr [, editmask], position [, options] [, summary]
```

Where:

level	Is the level where the item is to be printed. Each AskPlus report may be composed of a combination of different levels. For example, lines printed at the top of each page use the header (H) level. Lines printed at the beginning of a sort break use the begin sort or group (G) level. See *Report Levels* below for more information.
expr	Is the expression to be printed. This can be any expression supported by AskPlus. The expression is evaluated and the value is printed/displayed at the specified position, using the specified edit mask.
editmask	If specified, the editmask determines the specific formatting to be used. For example, a numeric edit mask may be used to display a numeric amount using a currency format. Another format might display a date field or a phone number. For more information see the EDIT statement.
position	Is the position on the specified line within the specified level where the item is to be printed. In most cases, the print position is required. See *Print Positions* below for more information.
options	A variety of print options may be specified to control how the expression will be printed. See *Print Options* below for more information.

Report Levels:

Each print statement must start by specifying the report level. Each level is specified by a one or two letter code shown below. The print statements may appear in any order, however, they will be printed in a pre-determined order as described below.

First (Fnn)

'F' lines are printed once on the **First Page** of the report. Information printed on the first page usually involves general information like title, date, author, selection conditions, etc. First Page lines are always printed, even when the current select file is empty. First lines are **not** printed if an Oracle selection returns zero rows.

Use *nn* to specify the line number within the level.

Since 'F' lines are printed even when no data has been selected, a common technique is to print a message on one or more 'F' lines using an IF condition. For example:

```
F1, "No data selected!", 1 left, if $ENTRIES = 0
```

Header (Hnn)

'H' lines are printed at the top of every page (except for the page generated by 'F' statements).

Use *nn* to specify the line number within the level.

Group (Gll/nn)

'G' lines are printed at the beginning of the sort break that matches the specified sort level (ll). The sort level is determined by the level (ll) that follows the G. For example, the level G3 will be printed at the beginning of the sort break specified by the S3 sort statement.

Since the highest numbered sorts have the highest priority, the 'G' levels will be printed in descending order. The G levels with the highest number will be printed first and the G1 level will be printed last.

Use *nn* to specify the line number within the level. For consistency with levels, 'G' lines are also printed in descending order, with the highest numbered lines printed first.

Detail (Dnn)

'D' lines are printed for each extended entry selected.

Use *nn* to specify the line number within the level.

Total (Tll/nn)

'T' lines are printed at the end of the sort break that matches the specified sort level (ll). The sort level is determined by the level (ll) that follows the T. For example, the level T3 will be printed at the end of the sort break specified by the S3 sort statement.

'T' levels are printed in ascending order with the lowest 'T' level being printed first.

Use *nn* to specify the line number within the level.

Final Total (TFnn)

'TF' lines are printed once at the end of the report.

Use *nn* to specify the line number within the level.

Trailer (Znn)

'Z' lines are printed at the bottom or every page. Space is always reserved at the bottom of each page to print the trailer lines. See also the DYNAMICTRAILER print option.

Use *nn* to specify the line number within the level.

Sort Levels and Line Numbers:

Sort Level:

When using the Begin Sort (G) and End Sort (T) print levels, you must specify both a sort level and a line number. The level determines which sort break to associate with the print statement.

ll The sort level (ll) is required for both the 'G' and 'T' statements. It represents the corresponding sort break (sort level). The 'G' levels are printed in descending order (highest priority sort first) and the 'T' lines are printed in ascending order (highest priority sort last). Contiguous numbers are not necessary but the level number must correspond exactly to those used in the sort statement.

If this number is not specified, the line number *nn*, described below, will be broken down into two digits: the first digit is considered as the sort number and the second digit as the actual line number. This syntax is recognized for the purpose of compatibility with ASK Version C. It is possible only if less than ten sort levels are used and less than ten printed lines per sort level. For example, G12 is interpreted as level 1, line number 2. If only one number is specified, the number is assumed to be the sort level and the line number is assumed to be 1.

Line Number:

All print statements should include a line number which determines the relative position of the print statement within the specified level. If no line number is specified, the line number is assumed to be 1.

nn	The line number (nn) is usually specified for each type of statement. For most levels, you can define up to 100 lines numbered from 0 to 99. (Detail line numbers can be up to 999.) A check is made to make sure that the total number of lines to be printed on a given page does not exceed the logical size defined for the page. If the line number is zero (0), the line will be combined with the last line of the previous level group (if there is one). For all levels, except 'G', line numbers are printed in ascending order F1, F2, ... For 'G' levels, the line numbers are printed in descending order. If no line number is specified, line number 1 is assumed.

Print Positions:

The print position is determined according to the following syntax:

```
{ pos [LEFT|RIGHT|CENTER] [JUSTIFY]    }
{ TABnn  [JUSTIFY]                     }
{ X                                    }
```

Where:

pos	is the numeric print position (column number). The print position must be a number between 1 and the maximum width of the report. The maximum width may be specified by the WIDTH statement. If no width is specified, a default width is determined by the OUTPUT statement.
LEFT	this keyword means that the print position is computed from the **left** of the item or expression to be printed. In effect, this keyword means "print **starting at** this position". If the column number of the position is not specified, justification is from the left-hand margin.
RIGHT	this keyword means that the print position is computed from the **right** of the item or expression to be printed. In effect, this specification means "print **up to** this position". If the column number of the position is not specified, justification is up to the right hand margin. This is the default.
CENTER	this keyword means that the print position is computed from

the **center** of the item or expression to be printed. In effect, this keyword means "print the same amount of characters to the right and left of this position". If the column number of the position is not specified, centering is done between the left and right margins.

JUSTIFY	this keyword means that leading and trailing blanks should be ignored (removed) when determining the print position.
TAB*nn*	Uses a calculated position (tab) as the print position. In most cases, tab positions are calculated using the width of the largest item printed at that tab number. Tabs positions can also be explicitly specified by the TAB statement. See the TAB statement for further details on use. AskPlus does not use actual TAB characters, it simply calculates the appropriate column number based on the width of the fields being printed.
X	The 'X' character at the print position means that the expression to be printed is actually a call to a programmed procedure (user exit) according to ASK Version C syntax. This feature is provided only for compatibility with procedures already written. In this case, the expression has the following format: '*name*(*parameter*)'.

Print Options:

You can request a certain number of options for each print line. They can be specified in any order, separated by commas. The different possible options are as follows:

```
[, IF cond                              ]
[, E{nn|Z}                              ]
[, SKIP  {A|B}                          ]
[, SPACE {A|B}nn                        ]
[, { ALL | MERGE | REPEAT [FIRST] }     ]
```

Where:

IF **cond**	Specifies that the expression described by this statement should be printed conditionally. The expression will be printed only if the condition is TRUE. If the condition contains any commas, the expression must be enclosed in a set of parentheses. If the parentheses are omitted, the commas will be mis-interpreted as separators for the print statement. For example:

```
D1, expr, pos, IF (ITEM = val1, val2)
```

E{nn|Z} Specifies an edit mask to use for the expression to be printed. The guidelines for edit masks are described in the EDIT statement. This option refers to an edit mask that is defined elsewhere. The edit mask can also be entered as part of the print statement. In this case, it must be entered **immediately before** the print position.

- Enn Enn, refers to the number of an edit mask defined by the 'E' (EDIT) statement.

- EZ EZ is a pre-defined (standard) edit mask used to correctly print data in *packed* or *zoned* format. It does not refer to an 'E' (`EDIT`) statement.

SKIP { A | B }

Specifies that a **page** skip is to occur **after** (A) or **before** (B) the line containing the statement is executed. This option will not be executed if a page skip has already occurred because the current page was full or if there was a page change caused by a `NOSPLITxxx` option or if it is requested within a header (H) or trailer (Z) statement.

SPACE { A | B }nn

Specifies that *nn* **lines** must be skipped **after** (A) or **before** (B) the line containing the statement is executed. If several statements pertaining to the same print line have different line skip options, AskPlus selects the **largest** value; under no circumstances are the values added together. Space is not continued after a page break.

ALL Indicates that the expression described must be repeated (printed again) for each entry of a secondary set if an item in the expression to be printed belongs to this secondary set and if one or more entries of the secondary set can be accessed for each extended entry in the select file. If several secondary sets are involved for different expressions with the `ALL` option, each one is taken successively (separate loops) and printed in sequence.

MERGE Performs a similar algorithm on the secondary sets as the `ALL` option, but the different sets are taken simultaneously (merged loops) and therefore are printed out in parallel.

REPEAT [FIRST]

Synchronizes the printing of data from the primary entry (or constants) with chains of entries containing the `ALL` or `MERGE` options. If the `FIRST` keyword is specified, only the first entry of each chain will be accompanied by the expression printed with `REPEAT`.

These options (ALL, MERGE, REPEAT) can be more easily understood by reviewing the examples later in this section.

Summary Options:

These options are used to print summary values for the current group. Summary options are valid only on 'T' or 'TF'' statements. The following options are available:

```
, { ADD | COUNT | AVERAGE | MIN | MAX } [ALL]
```

Where:

ADD	Total of the expression for the current group. This option is only valid for numeric data.
COUNT	Count the number of entries belonging to the current group. This option is valid on any type of data.
AVERAGE	Average of the expression for the current group. This option is only valid for numeric data. The Average is calculated as the total divided by the count. Weighted averages must be calculated manually using registers.
MIN	The minimum value of the expression for the current group. This option is valid for numeric and character data.
MAX	The maximum value of the expression for the current group. This option is valid for numeric and character data.

These summary values are calculated automatically by AskPlus using unnamed registers. The values may be printed, but cannot be used in other calculations. These options are only valid on 'T' and 'TF' statements. If the `ALL` option is not specified, only the extended entries (entries saved in the select file) are used to calculate the summary values. When ALL is specified, all secondary chains are included in the calculation. See the examples below for more information.

Special Form of Print Statement (Reset)

A special form of the print statements is available for resetting a *variable* or for forcing a *flag* to be set **after** the line corresponding to this statement is printed. This syntax is primarily used for compatibility with Ask Version C syntax. This special form has the following syntax:

```
{level},   { variable          } [, IF condition ]
           { SET|CLEAR flag    }
           { $PAGENO           }
```

It is important to understand that compute (#variable) and flag (`IF` statement) statements are executed **before** the print statements, whereas these special forms are executed **after** the print statement for a each extended entry.

Where:

level	Is the level where the reset statement should be executed. See the previous section for more information on levels.
variable	Is the name of a the variable to be reset. Numeric registers are reset to zero character registers are reset to blanks and boolean (K1) variables are set to $FALSE.
SET *flag*	Sets the specified flag (see IF statement) to TRUE.
CLEAR *flag*	Sets the specified flag (see IF statement) to FALSE.
$PAGENO	Resets the current page number to 1. This statement is usually accompanied by a SKIP option. For example Tnn, $PAGENO,,SKIP A In the previous example, 2 commas are needed between $PAGENO and SKIP. Normally the print position would be placed between the commas. In this special case, no print position is needed, however, the comma is still required.
condition	When an IF condition is specified. The reset command will only be executed when the condition is TRUE.

4.16 EDIT Statement [E]

Specify an edit mask or format. The edit statement is used to define an edit mask that will be used by one or more print statements using the 'Enn' option of the print statement.

```
Enn, "format"
```

Where:

nn	Edit mask number. This number is used for all references to this format in print statements.
"format"	Character string representing the format to be used. Different characters have different meaning depending on whether the value to be printed is numeric or character. See the discussion of formats that follows.

Discussion:

- AskPlus uses edit masks (also called formats) to format how values are printed. Formats are most commonly used for numeric fields, however, character formats are also available.
- Formats created via the EDIT statement are referenced by their number (Enn) in print statements. More than one print statement may reference the same edit mask.
- Edit masks can be entered directly in the print statement without using the Edit statement.
- If the number of significant digits in the value to be printed is greater than the number of position characters in the edit mask, the formatted zone is filled with asterisks.

Numeric Formats:

The easiest way to understand formats, is to look at some examples. The following table shows how several numbers would be printed by the following formats:

Format	**Value to print**	**Result**
"$$$,$$9.99-"	123456	$1,234.56
	0	$0.00
"zzz,zz9.99-"	-123456	1,234.56-

	0	0.00
"zzz,zz9!99-"	123456	123,456.00
	0	0.00

How They Work:

A numeric format consists of *position* characters, *alignment* characters, *sign* characters and *insertion* characters.

Position Characters:

Position characters are used to signify the positions of individual digits to be printed. All of the position characters represent the position of a digit, but with a different emphasis. The position characters are: 9 ^ Z * $, and have the following meaning:

9 or ^	Represents the position of a digit. If the digit is zero, the "0" will be printed. This includes non-significant zeros at the beginning of the number.
Z (Zero)	Represents the position of a digit. If the digit is a non-significant zero (at the beginning of a number) it is replaced by a blank.
*	Same function as 'Z' except that non-significant zeroes are replaced by asterisks.
$	Replaces non-significant zeroes by blanks except for the last (right most) which is replaced by a dollar sign $.

If the number of significant digits in the value to be printed is greater than the number of position characters in the format, the formatted zone is filled with asterisks.

Alignment Characters:

Alignment characters are used to align the decimal point of the number. There are two alignment characters:

.	When the decimal point is used in a numeric format, the position of the decimal point in the formatted data is determined by the number of position characters that appear to the right of the decimal point. Use the decimal point when printing integer (whole number) values. For example, when printing the value 123 with the mask "9.99", the result is 1.23.

	If the mask is "9.999", then "0.123" will be displayed.
!	When the exclamation point (!) is used, the decimal point in the displayed value is aligned with the actual decimal point in the value to be displayed. Use this character when you want to control the number of decimals displayed for real (floating point numbers) numbers. For example, when printing the value 123.45 with the mask "999!99", the result is "123.45". If the mask is "999!999", then "123.450" will be displayed.

Sign Characters:

Sign characters are used to display the sign of negative numbers. The sign characters are as follows: - CR DR

-	The '-' (minus sign) represents the sign **only** if it is placed in the left-most or right-most position in the format and will be printed either to the left of the first significant digit in the number or in the last position to the right of the number, only if this number is negative. If the character appears in another position within the format, it is treated as an insertion character (see below).
CR	Means CREDIT. The 'CR' is only considered a sign character if it appears in the right-most position of the format. The CR will be printed along with the value only if the number is negative.
DR	Means DEBIT. Operates in the same way as CR. The DR will only be printed if the value of the number is negative.

WARNING: *If **no sign character** is included in the mask, negative numbers will **appear** to be positive numbers because the sign will **not** be displayed.*

Insertion Characters:

All other characters are treated as *insertion* characters and are therefore inserted directly at the corresponding position in the formatted zone.

Examples:

Value	**Format**	**Result**	**Comments**
`123456`	`"999,999.99-"`	`001,234.56`	Display leading zeros.
`123456`	`"zzz,zz9.99-"`	`  1,234.56`	No leading zeros.
`123456`	`"99999"`	`*****`	Format too small.
`123456-`	`"$$$,$$9.99-"`	` $1,234.56-`	Leading $ sign.
`123456-`	`"***,**9.99-"`	`*$1,234.56-`	Leading * and $ sign.
`123.456`	`"zz9!999"`	`    123.46`	Use the ! to mark the decimal.
`123.456`	`"zz9!9"`	`    123.5`	Round the decimal point.
`123456`	`"999-999"`	`123-456`	"-" is an insertion character.

Character Formats

An format for character values consists of a string of characters formed by the *position character* 'X' and *insertion characters*. Each occurrence of the 'X' character represents the position of a character of the value to be printed. All other characters are insertion characters and are replicated in the formatted string.

If the format has fewer position characters (nn) than the number of characters in the value to be printed, the formatted zone will contain only the first *nn* characters of the original value.

Examples:

Value	**Format**	**Result**
YYYYMMDD	"xxxx/xx/xx"	YYYY/MM/DD
8008482565	(xxx) xxx-xxxx	(800) 848-2576
982901290	xxxxx-xxxx	98290-1290
ABCD	x-x-x-x	A-B-C-D
ABCD	(xxx)	(ABC)

4.17 Understanding Merge, All & Repeat

The MERGE, ALL and REPEAT are used to tell AskPlus how to handle fields in secondary datasets or files. All of these options are available in the REPORT command. The MERGE and REPEAT options are available in the SAVE command. Before explaining how these options work, it is first necessary to understand the differences between a primary and secondary set (or file) in AskPlus.

What is a Primary Set?

When AskPlus executes the FIND command, it creates a select file that contains information about all of the records that were selected. The OUTSEL statement, which is part of the FIND, determines which datasets and files will be included in the select file. During the FIND command, the PRIMARY statement determines which set is the PRIMARY set. During the REPORT (or SAVE) commands, **all of the datasets and files in the select file are considered PRIMARY sets because they are part of the extended entry.**

In most FIND commands (unless the INSEL statement is used), there is only one PRIMARY set. However, in the REPORT (or SAVE) commands, the primary is determined by the current select file. This means that during the REPORT (or SAVE), there may be more than one PRIMARY set. This is often called an extended entry.

When AskPlus processes records from the primary, each record (extended entry) is processed exactly once.

What is a Secondary Set?

Any dataset or file that is used in the REPORT or SAVE command, **but is not included in the select file**, is a secondary file.

Note: This is not the same as a VISIMAGE Secondary Topic. A secondary topic in VISIMAGE is **usually** included in the select file and therefore at REPORT/SAVE time it is no longer a secondary, but is now part of the extended entry.

By default, when AskPlus processes records from a secondary set (or file), only the first record in the secondary set will be processed. However, if the secondary set has multiple records, the MERGE and ALL options may be used to process all of the records in the secondary set.

What does Merge do?

When a field from a secondary is output using the MERGE option, AskPlus will automatically process all values from all records in the secondary that match the appropriate key value. In other words, AskPlus will read all the corresponding detail records and include them in the output. If MERGE is not specified, only the value from the first record in the chain will be output. The MERGE loop is executed at the same time as the extended entry is processed. If fields from two different secondaries are printed using the MERGE option, the chains will be processed at the same time and fields (as much as possible) will be printed side by side. The MERGE option is valid for both the print statement and the compute (variable) statement.

What does All do?

The ALL option is similar to the MERGE option. Like MERGE, the ALL option causes AskPlus to print all of the related entries in the secondary set, however, unlike MERGE, entries from separate secondary sets will be processed in separate loops. Fields processed with the ALL option are processed **after** the extended entry is processed and **after** the MERGE loop (if any) is processed. If fields from two different secondaries are printed using the ALL option, separate ALL loops will be executed for each set. Like MERGE, the ALL option is valid on the print and compute (variable) statements.

What does Repeat Do?

When a field from a **primary record** is output using the REPEAT option, ASKPLUS will **repeat** the same primary value on all additional records that are output as the result of the MERGE or ALL loops that occur elsewhere. In other words, if AskPlus prints any additional records because of the MERGE or ALL options, fields that contain the REPEAT option will be repeated.

Example of MERGE & REPEAT

Lets look at an example using a credit union database. We will use three datasets: MEMBER-FILE (list of members), SHARE-FILE (list of accounts) and LOAN-FILE (list of loans).

In all of the examples, MEMBER-FILE is the only primary set; both SHARE-FILE and LOAN-FILE are secondaries. These examples use REPORT to demonstrate MERGE and REPEAT because it is easier to see the results. The last example uses SAVE.

Example #1 - No Special Options

```
FIND
PRIMARY=MEMBER-FILE
ACCOUNT=7297, 11677, 18774
END
```

```
REPORT
D1, ACCOUNT, TAB1
D1, SHARE-FILE.BALANCE, TAB2
D1, LOAN-FILE.BALANCE, TAB3
END
```

Account	Share Balance	Loan Balance
7297	$22,612.35	$410.43
11677	$29,706.29	$5,238.14
18774	$22,535.59	$217.31

Notice that **only one line** is printed for each account, even though the member may have multiple share and/or loan accounts. This is because no MERGE or REPEAT option was specified.

Example #2 - Include Merge

```
REPORT
D1, ACCOUNT, TAB1
D1, SHARE-FILE.BALANCE, TAB2, MERGE
D1, LOAN-FILE.BALANCE, TAB3, MERGE
END
```

Account	Share Balance	Loan Balance
7297	$22,612.35	$410.43
	$4,428.86	$0.00
		$5,008.18
		$0.00
11677	$29,706.29	$5,238.14
	$3,747.95	$2,816.25
	$781.37	
	$1,040.07	

18774	$22,535.59	$217.31
	$8,308.46	

This time, because MERGE was specified, we see all of the share and loan accounts (even if the BALANCE is zero.)

Example #3 - Include Repeat

```
REPORT
D1, ACCOUNT, TAB1, REPEAT
D1, SHARE-FILE.BALANCE, TAB2, MERGE
D1, LOAN-FILE.BALANCE, TAB3, MERGE
END
```

Account	Share Balance	Loan Balance
7297	$22,612.35	$410.43
7297	$4,428.86	$0.00
7297		$5,008.18
7297		$0.00
11677	$29,706.29	$5,238.14
11677	$3,747.95	$2,816.25
11677	$781.37	
11677	$1,040.07	
18774	$22,535.59	$217.31
18774	$8,308.46	

The results are similar to example 2 except that the account number is repeated for every line.

Example of MERGE and REPEAT with SAVE

The last example uses the SAVE command to create an SD file. Since we are printing two items with the same name (BALANCE) we have to use registers to assign them a new name in the file.

```
SAVE OUT=TSTMERGE,SD
NEWREG SHAREBAL:I2
NEWREG LOANBAL:I2
#SHAREBAL=SHARE-FILE.BALANCE,MERGE
```

```
#LOANBAL=LOAN-FILE.BALANCE,MERGE
D1, ACCOUNT, TAB1, REPEAT
D1, #SHAREBAL, TAB2, REPEAT
D1, #LOANBAL, TAB3, REPEAT
END
```

This creates a file with the following data:

Account	SHAREBAL	LOANBAL
7297	2261235	41043
7297	442886	0
7297	0	500818
7297	0	0
11677	2970629	523814
11677	374795	281625
11677	78137	0
11677	104007	0
18774	2253559	21731
18774	830846	0

Note: This is the same as the previous example, except that blanks are replaced by zero and there is no edit mask.

In most cases it is important to include REPEAT when using the SAVE command because you need the value (in this case ACCOUNT) repeated in order to identify the record.

Also note that the compute statements use MERGE and the print statements for the registers use REPEAT. This is because MERGE was already used when the registers were calculated.

4.18 REPORT ALL Command [R ALL]

Print a listing that includes all of the fields from the dataset(s) in the current select file. Each item in each set is printed, one item per line.

```
REPORT ALL [, Q[UIET] ]
```

Discussion:

This special form of the REPORT command means that all the accessible items in the set(s) of the current select file will be displayed (or printed out) in simplified format: each item is displayed with its name, one item per line and with a line feed (blank line) between each record.

If the QUIET option is used, the item names are **not** displayed and there is no blank line between each record. The QUIET form of the "R ALL" command is very useful to create a backup file prior to deleting records. See examples below:

Examples :

This example selects 3 records and displays the data using "r all":

```
>find limit=3;memo-file
Current OUTPUT select file is ./ASK66301
using serial read
entries read 4 4
entries qualified 3

>r all
using previous select file
ACCOUNT                   = 604
SUFFIX                    = 0
NUMBER                    = 600
ENTRY-DATE                = 19980406
MEMO                      = Memo 1
EXPIRATION-DATE           = 0
UPDATE-COUNTER            = 1

ACCOUNT                   = 603
SUFFIX                    = 0
NUMBER                    = 600
ENTRY-DATE                = 19980406
MEMO                      = Memo 2
EXPIRATION-DATE           = 0
```

```
UPDATE-COUNTER         = 1

ACCOUNT                = 602
SUFFIX                 = 0
NUMBER                 = 600
ENTRY-DATE             = 19980406
MEMO                   = Memo 3
EXPIRATION-DATE        = 0
UPDATE-COUNTER         = 1
```

The "REPORT ALL" command displays the data, one field per line with the item name preceding the item value.

In this example, we use the "Quiet" option to create a backup file:

```
>f memo-file.number=600
Current OUTPUT select file is ./ASK66301
using serial read
entries read 2768 2768
entries qualified 28

>out=MEMOBKUP
>r all, quiet
using previous select file
>out=term

>delete
entries to be deleted. OK to proceed [no] ? yes
```

In this example, the file MEMOBKUP contains all of the data in the selected records prior to the delete. If necessary, this file will be available to add the data back into the database. To use the file for an ADD, two lines must be added to the file, one at the beginning and one at the end.

```
First Line: ADD MEMO-FILE
Last Line:  //
```

Now, the file is ready to use for an add, as follows:

```
>xeq MEMOBKUP
```

5 Creating Data Files

5.1 Overview

In addition to creating ASCII output using the REPORT command, AskPlus is able to create (binary) file output using the SAVE command. The SAVE command is used to create data files that will (in most cases) be used by other programs. The SAVE command is able to create a wide variety of file formats.

Both the REPORT and SAVE commands are capable of creating files, but there are two important differences between these commands:

- A file created by the REPORT command will always contain formatted ASCII data. The contents of the file will be identical to the same report when it is printed. In contrast, the SAVE command will create the file using the same data type as the original field. This could be ASCII or some other binary type like I2 or P20.
- A file created by the REPORT command can include data from different print levels (H, Gnn, Dnn, ...). In contrast the SAVE command will only output a single level when it creates the file.

In most cases, the SAVE command will be used if you want to create a file that will be used by another program. Use the report command, if you want to create a file that will be read by people.

The SAVE command is very similar to the REPORT command. This chapter will list the statements that are supported by the SAVE command and discuss how these statements differ from the equivalent statement in the REPORT command. Like a report, a save file is created with data extracted from a previously opened domain. See ***Chapter 2, Accessing Data*** (Page 19) for information on opening and closing domains. Except when using Oracle, the data to be included in the file should already have been selected by a previous FIND command. See Chapter 3, Data Selection for more information on selecting data. Use the CHOOSE statement (see below) to select data for a file using an Oracle database.

Command Structure

Like the REPORT command, a SAVE command consists of a series of statements terminated by the 'END' statement. Taken together, these statements determine what fields will be included in the file, how the file will be formatted and where the file will be created. These statements fall into the following classifications.

Output and Control Statements

These statements are used to specify certain global options and to control what type of file will be created.

Data Selection Statements

These statements determine the source of the data that will be used to create the file. Unless you are working with Oracle, these statements are not normally required. By default, AskPlus will use the data selected during the most recent FIND command as the data source.

Variable and Compute Statements

These statements allow you to create variables and specify calculations using the variables.

Print and Formatting Statements

These statements control exactly what data will be included in the file and how it will be sorted.

Save ALL

A special form of the SAVE command, called 'SAVE ALL', provides a quick way to create a file that includes all of the data fields referenced in the current select file. When this form of the command is used, only the OUTPUT statement is allowed. This command is very useful for creating a backup file prior to deleting data records.

5.2 SAVE Command [SA]

Create a data file in a specified format.

```
SAVE
OUT = filename [, option ] [;]
statement [;]
...
END
```

The SAVE command uses essentially the same statements and syntax as the REPORT command. This chapter will document the areas in which those statements differ from the REPORT command. Refer to *Chapter 4, Reporting Data* for more information on the REPORT command.

Remember that the REPORT command should be used to create readable (formatted) ASCII files and the SAVE command should be used to create data files that will be used by other programs. (It is, of course, possible for other programs to read files created by the REPORT command, but this is not its primary purpose.)

The files created by the SAVE command will often be used by AskPlus when creating multi-pass reports.

Output and Control

These statements are used to specify the type of file being created and its name as well as some global settings..

OUTPUT	Specifies the name and type of file being created. The options for this command are different from the REPORT command.
Control	Only a few of the control statements are valid in the SAVE command.
INCLUDE	Specifies the name of a command file to be included. Identical to the Report command.

Data Selection

These statements are used to specify the data source to be used for the file. These statements are necessary for Oracle databases, but are not normally used with other databases and files. By default, the current select file will be used as the data source for the file. The data can be limited with the LIMIT statement and further refined using the CHOOSE statement.

These statements are identical to the REPORT command. Refer to the previous chapter for more information.

PRIMARY	Specifies the primary dataset to be used.
INSEL	Specifies the name of the input select file.
CHOOSE	Refines the current selection.
LIMIT	Limits the number of records used in the report.
LINK	Specifies an access path to other datasets or indexed files.

Variable/Compute

These statements are used to create temporary variables (registers) that will be used in the current command and also to specify the calculations to be performed with the variables.

These statements are identical to the REPORT command. Refer to the previous chapter for more information.

NEWREG	Creates a new variable (register).
#variable	Specifies a compute statement.
IF	Calculates the value of an ASK Version C flag. This statement has been replaced by the more general compute statement.

Print/Formatting

These statements specify how the file will be sorted and what data fields will be included.

SORT	Specifies the sort order. Identical to the Report command.
Print	Specifies how each field should be printed. Similar to the REPORT command, but several differences will be discussed in this chapter.
EDIT	Specifies an edit mask. This statement is not used by the SAVE command. Data in the file is **not** formatted, unless the !FORMAT function is used.
END	This keyword specifies the end of the statements in the SAVE command. Except in the special case of "SAVE ALL", the END statement is required. It marks the end of the syntax analysis phase of statements and means that the save logic will

be checked and the actual file generated.

In the remainder of this chapter, each of these statements will be discussed in more detail in the order that they appear above.

5.3 OUTPUT Statement [OUT]

The form of the `OUTPUT` statement is :

```
OUT[PUT] [=] filename[, type] [, option ]
```

An `OUT` statement is always required to specify the name and type of the file being created by this command.

Where:

filename	is the name of the file being created by this command. The file name is required. On MPE systems, the file name must use MPE syntax (`FILE[.GROUP[.ACCOUNT]]`). POSIX syntax is not supported. On UNIX systems, any valid filename may be used.
type	is the type of the file being created - see *File Types* below.
option	is one or more options - see *File Options* below.

File Types:

AskPlus supports a wide range of file types. Use the type that is best suited for the application that will use the file.

FLAT The file is produced in simple binary sequential format, each print statement (item) creates a separate column in the file. The file contains fixed length records with no special formatting. This is the type of file that will be created, if no type is specified in the OUTPUT statement.

On UNIX systems, no 'newline' characters will be added at the end of each record. Any program reading this file will have to know the exact file layout and the length of each record. This file type is useful for adding data into a database using SUPRTOOL's PUT command. See "SAVE ALL" for more information.

SD The file produced is organized in SD (self describing) format. An SD file is a file that contains information about the data contained in the file. We refer to this information as the data header which precedes the actual data. Unlike FLAT files, programs that understand the structure of an SD file will be able to read the file and "know" what fields have been saved in the file and the length of each record. The actual structure of an SD file varies between systems.

On MPE systems, the SD files created by AskPlus are identical to LINK files created by SUPRTOOL. Both AskPlus and SUPRTOOL are able to exchange information using this SD format. On MPE systems, the data header is written in the file label and the actual data portion of the file begins in record zero.

On UNIX systems, the data header created by AskPlus is similar to, but not identical, to the header on MPE systems. On UNIX, an SD file consists of 2 separate files. The data header is stored in the one file (and has the same name as the name specified in the OUTPUT statement) and the data file is a separate file. The data file will have the same name as the header file, but with a ".dat" extension.

Note: SUPRTOOL LINK files created on UNIX also consist of two files, however, in SUPRTOOL the header file has a ".sd" extension and the data file has no extension. When opening a SUPRTOOL SD file on UNIX, you must specify the ".sd" extension as part of the filename.

ASD "c1 c2" This option will create an "ASCII Delimited" file. All fields are separated by the delimiter (c1) and character data is enclosed in quotes (c2). The default delimiter is a semi-colon, but can be changed to another value by enclosing the delimiter (c1) in quotes after the ASD keyword. To use a TAB as the delimiter specify ^t as the delimiter.

Numeric fields are not formatted, but may be formatted using !FORMAT. If they are formatted, they are treated as character (non-numeric fields.)

Character fields will be enclosed in quotes (or its replacement 'c2').The default quote character, which is added around character data, is the double quote. It can also be changed by specifying a new character in the string that follows the ASD keyword. (See the examples below.)

On UNIX the ASCII delimited file will have a ".asd" extension.

WKS Creates a WKS formatted file which can be read by either LOTUS$^{(Tm)}$ or EXCEL$^{(Tm)}$.

SYLK Creates a SLYK formatted file which can be read by EXCEL$^{(Tm)}$ and other Microsoft products.

WDP Creates a formatted file which can be read by WORDPERFECT$^{(Tm)}$.

DIF Creates a DIF (Data Interchange Format) format which is used by some PC programs.

File Options:

INDEXED ([*keyfile*][:*size*])

Both FLAT and SD files can also be created as INDEXED files. The advantage of an indexed file is that AskPlus can link to this file in a multi-pass report. If a file is not indexed, it can be used as a primary file in a subsequent report, but **not** as a secondary file.

On MPE systems an indexed file is created as a KSAM file. On UNIX systems they are created as C-ISAM files. C-ISAM must be available to create indexed files on UNIX.

At least one print statement must contain the key option when creating an indexed file.

For backwards compatibility with MPE-V, the name of the KSAM keyfile and its size can be specified. In most cases, this parameter is blank, however you must include the parentheses. "INDEXED ()"

DELETE When this option is specified, the file (if it exists) will be automatically deleted before the new file is created. If DELETE (and APPEND) are **not** specified, AskPlus will prompt for permission to delete the file before proceeding. In batch, the file will be automatically deleted.

APPEND When APPEND is specified, AskPlus will append output from the current file to the end of an existing file of the same name. It is the user's responsibility to ensure that the structure of the file being appended is identical to the existing file. When using APPEND, you should use the same file type that was used when the file was originally created. If the file does not exist, it is created. The APPEND option is only valid on FLAT and SD file types.

OPEN If the OPEN option is specified, the created file is automatically opened as a new domain and is immediately available for use with future AskPlus commands.

Examples:

Creating SD files:

```
OUT=FILE.GROUP, SD, DELETE  <<MPE>>
```

```
OUT=../DIR1/FILE, SD, DELETE <<UNIX>>
OUT=MYFILE, SD, INDEXED(), DELETE
OUT=MYFILE, SD, INDEXED(), APPEND
```

Creating ASD files

```
OUT=FILE, ASD ",'", DELETE
```

Creates a file with a comma as the delimiter and a single quote as the character to surround the character fields.

```
OUT=FILE, ASD "^t", DELETE
```

Creates a TAB delimited file that uses double quotes (default) around the character data.

Note: The ASCII delimited files on UNIX are created with a ".asd" extension.

5.4 Control Statements

Only a few of the REPORT control statements are valid in the SAVE command.

```
keyword [[=] parameters]
```

The following statements are available in the SAVE command to define execution parameters.

COMMERCIAL	When using this option, AskPlus default numeric variables will be created using an numeric (P20) type. This option cannot be used with the SCIENTIFIC option.
SCIENTIFIC	When using this option, AskPlus default numeric variables will be created using a floating point numeric type. This option cannot be used with the COMMERCIAL option.
ROUND	Causes AskPlus to treat the "/" division operator like the DIVR and return a rounded result. By default, the result of a division using "/" will be truncated and "DIVR" will be rounded. This option is not recommended, since many calculations assume that the result will be truncated. Use DIVR instead if you wish to force a rounded result.

If any invalid REPORT keywords are specified in the SAVE they will generally be ignored. In most cases, an error will not be printed. This makes it much easier to convert an existing report into a save without forcing you to delete the unnecessary statements.

5.5 Print Statements

Print statements are used to print each element of the file. Each print statement specifies the item or expression to include along with its relative position in the file.

Each print statement has the form:

`level,` *expr, position* `[,` *options*`] [,` *summary*`]`

Where:

level	Is the level where the item is to be printed. The level is identical to the level in the REPORT command. See *Report Levels* in the previous chapter for more information. The only levels which are valid in the SAVE command are G*ll*, D, T*ll* and TF. Any other levels (F, H and Z) are ignored. Only **one** level is permitted in **each** SAVE command. If more than one level is specified, the lowest (most detailed level) will be used. If there is a detail level, it will be used. If there is no detail, the lowest End Sort (T*ll*) level will be used. The Begin Sort level is valid, but is rarely used.
expr	Is the expression to be included in the file at that position. The expression is evaluated and the value is included in the file at the specified position. This can be any expression supported by AskPlus, but is usually either an item or a variable. Using an expression, other than an item name or variable, particularly with SD files, is **not** recommended. Rather than using an expression, use a variable and load the value of the variable with the expression.
position	The print position is required syntactically, but its value is ignored. An exact position is not necessary, since the SAVE command places each field next to the previous field with no intervening blanks. The first field starts at the beginning of the record and it is followed immediately by the second field. Use (tab1, tab2, tab3), or (1, 2, 3) for the print position. AskPlus will include the fields in the file in the order of the print statements, the actual print position is not used. Except for a few exceptions, the print position is required.
options	A variety of print options may be specified to control how the expression will be handled. See *Print Options* below for more information.

Report Levels:

Each print statement must start by specifying the report level. Each level is specified by a one or two letter code shown below. The print statements may appear in any order, however, they will be printed in a pre-determined order as described below.

- Only the levels G*ll*, D, T*ll* and TF are valid.
- If G*ll* or T*ll* is used, the Sort level (nn) must match an existing SORT statement.
- Only one level is allowed in each SAVE, the most detailed level will be used, if more than one level is specified.
- The level does not need a line number, multiple lines are not permitted.

Print Options:

You can request a certain number of options for each print line. They can be specified in any order, separated by commas. The different possible options are as follows:

```
[, KEY                      ]
[, IF cond                  ]
[, {MERGE | REPEAT}  ]
```

Where:

KEY	Specifies that this column will be a key field in the indexed file. This option is only valid when INDEXED has been specified on the OUTPUT statement. At least one KEY field must be specified for an indexed file.
IF **cond**	Specifies that the expression described by this statement should be included conditionally. The expression will be included in the file only if the condition is TRUE. If the condition contains any commas, the expression must be enclosed in a set of parentheses. If the parentheses are omitted, the commas will be mis-interpreted as separators for the print statement. For example:

```
D1, item, pos, IF (ITEM = val1, val2)
```

When using an IF condition in the SAVE command, you **must** use the **same** IF condition with all of the print statements. Otherwise, the file will not be useable because the structure of each of the records will not be the same.

	Use variables and compute statements to change the value of a single column.
MERGE	Indicates that the expression described must be repeated (included again) for each entry of the secondary set if an item in the expression belongs to this secondary set and if one or more entries of the secondary set can be accessed for each extended entry in the select file.
REPEAT	Synchronizes the printing of data from the primary entry (or constants) with chains of entries containing the `MERGE` option. If the `FIRST` keyword is specified, only the first entry of each chain will be accompanied by the expression printed with `REPEAT`.
	These options (ALL, MERGE, REPEAT) can be more easily understood by reviewing the examples in the previous chapter.

Summary Options:

These options are used to print summary values for the current group. Summary options are valid only on 'T' or 'TF'' statements. The following options are available:

```
, { ADD | COUNT | AVERAGE | MIN | MAX } [ALL]
```

Where:

ADD	Total of the expression for the current group. This option is only valid for numeric data.
COUNT	Count the number of entries belonging to the current group. This option is valid on any type of data.
AVERAGE	Average of the expression for the current group. This option is only valid for numeric data. The Average is calculated as the total divided by the count. Weighted averages must be calculated manually using registers.
MIN	The minimum value of the expression for the current group. This option is valid for numeric and character data.
MAX	The maximum value of the expression for the current group. This option is valid for numeric and character data.

These summary values are calculated automatically by AskPlus using unnamed registers. The values may be printed, but cannot be used in other calculations. These options are only valid on 'T' and 'TF' statements. If the `ALL` option is not specified, only the extended entries (entries saved in the select file) are used to calculate the summary values. When ALL is specified, all secondary chains are included in the calculation. See the examples in the previous chapter for more information.

5.6 SAVE ALL Command

Save **all** of the items of a set or a file to the specified file in the specified type.

```
SAVE
OUT = filename [, option ] [;]
ALL
```

Where:

filename	Is the name of the file being created.
option	Is one or more file output options.
ALL	Is a keyword that means ALL items from all sets saved in the current select file will be included in the output file.

The syntax for the OUT statement is described earlier in this chapter.

Discussion:

- ALL of the fields in the current select file will be included in the output file.
- This option is useful for creating a data file that will be used with the SUPRTOOL PUT command to add data to a dataset. It can be done as a backup prior to a delete, or to copy data from a production dataset to a test dataset.

Examples:

Creating a file to use with SUPRTOOL:

```
>FIND PRIMARY=MEMO-FILE;OUTSEL=(MEMO-FILE)
>>NUMBER = ?Enter_Memo_Number

* Create a backup file

>SAVE
>>OUT = MEMODATA, FLAT, DELETE
>>ALL
```

Adding data with SUPRTOOL:

The following code will add the data back into MEMO-FILE using SUPRTOOL. On UNIX, the reclen and the nolf (no line feed) parameters must be specified on the input statement. (These options are not required on MPE.) For MEMO-FILE, the reclen is 80 bytes. Use the "FORM SETS" command to find the record length (entry length) of a dataset.

On UNIX, the number of bytes in a file created by AskPlus using SAVE ALL and the FLAT option should equal the number of records multiplied by the entry length. No extra bytes will be added.

```
SUPRTOOL
base=membrs,1,password;
input MEMODATA, reclen 80, nolf <<UNIX>>
input MEMODATA                  <<MPE>>
put memo-file
xeq
exit
```

Creating an ASCII Delimited file:

```
>SAVE
>>OUT=MYASDFILE, ASD, DELETE
>>ALL
```

Note: The SAVE ALL command cannot generate indexed files, because there is no way to specify the key fields.

6 Updating Data

6.1 Overview

The primary purpose of AskPlus is data selection and reporting; however, AskPlus also provides commands that allow you to modify data. These commands do not replace transaction processing programs that are designed for data entry and modification. Instead they provide a quick and easy way to add test data, delete unnecessary records or update data values. Care must be taken when using these commands since the original data cannot be restored unless the database has been previously backed up. Please remember the following whenever you use any of these commands.

- These commands only work with TurboImage compatible databases. These include Image, TurboImage and Eloquence. Oracle databases accessed via the OpenTurbo interface are also supported.
- These commands do not work with Oracle databases accessed using the Oracle OCI, nor do they work with files.
- Be sure to carefully test your procedures using a test database *before* you make changes to a production database.
- You must have write access to the database to use these commands.
- To disable this commands, set the system variable PROTECT=1.

The following functions are available.

`ADD`	Adds entries to the *dataset*.
`REPLACE`	Replaces (updates) the values of the specified *fields* with a new *constant* value.
`UPDATE`	Updates the specified *field* using a *dynamically* calculated value. Updates are done using a special form of the variable compute statement.
`DELETE`	Deletes all of the selected records. The DELETE command will delete ALL of the *extended entries* in the current select file. Be very careful when using this command.

6.2 ADD Command [AD]

Add one or more entries to a *set*. In interactive mode, AskPlus will prompt the user for the values to be added. Default values can be added using the "USING filename" syntax. When used via the XEQ command, the data values are supplied by the XEQ file. See examples below.

```
ADD dataset [ USING filename ]
```

Where :

dataset	is the name of a *dataset* in one of the open *domains*.
filename	is the name of a QEDIT or EDITOR file. The format of a record is :

```
ITEM [(index)] = [ "value" [ = ] ]
```

Discussion:

When adding data, AskPlus will prompt for the value of the items to be added. For compound items, each *sub-item* will be requested separately. If '//' is entered while AskPlus is prompting for the value of a *sub-item*, no more *sub-items* will be requested and the next *item* will be prompted. If '//' is entered as the value of a *item*, the `ADD` will be terminated and the current record will not be added.

- If the USING parameter is not used, AskPlus will prompt for every item in the specified dataset.
- The "USING filename" parameter is used to specify default values for specific fields in the dataset. There are essentially 4 different ways that default values can be specified. Default values are specified as follows:

ITEMNAME =

When an item is entered but no value is specified, there is no default value. In this case, AskPlus will always prompt for the value to use for this item. The item name *must* be followed by the equal sign.

ITEMNAME = "value"

When a default value is provided, this value will always be used for the specified item. AskPlus will *not* prompt for a value. The default value *must* be enclosed in quotes.

ITEMNAME = "value" =

Adding a trailing equals sign combines the default value with a prompt.

AskPlus will prompt for a value. If return is pressed, the default value specified in the file will be used.

Item does not appear in the file

Values for fields *not* listed in the file will not be prompted during the add. Fields that are not listed in the file will default to a NULL value.

To add entries to a dataset you must have write access. If the *Job Control Word* `PROTECT` is equal to 1, this command will be rejected.

ADD Example

This example demonstrates using the ADD command to add records to MEMO-FILE. This dataset contains the items ACCOUNT, SUFFIX, NUMBER, ENTRY-DATE, MEMO, EXPIRATION-DATE and UPDATE-COUNTER.

The ADDMEMO file contains the following:

```
ACCOUNT =
SUFFIX = "50"
NUMBER = 100 =
ENTRY-DATE = "20050101"
MEMO = "This is the memo text"
```

Given this scenario, the ADD command works as follows:

```
>ADD MEMO-FILE USING ADDMEMO

ACCOUNT        =>>123456
NUMBER         =>>100

ACCOUNT        =>>123456
NUMBER         =>>100

ACCOUNT        =>>// << end of add command >>
```

Using this example, AskPlus will prompt for the values for the items ACCOUNT and NUMBER. The item NUMBER will default to "100" if enter is pressed. The values for ENTRY-DATE and MEMO default to the values specified in the file. The items EXPIRATION-DATE and UPDATE-COUNTER will default to zero.

ADD Example using XEQ

A common method of adding data, is to use AskPlus to create an XEQ file which contains both the ADD command and all of the data values that will be used to add the data. This is done using the REPORT command to create the XEQ file.

In the following example, dataset records are selected based on the value of a specific flag. The selected records are then used to populate the values that will be used to add the data. In this example, we will supply all of the values and will not use the "USING filename" syntax.

```
NEWREG ADDFILE:X100

MAKETEMP #ADDFILE

FIND SHARE-FILE.CONTROL-FLAGS(21) = 1

REPORT
LINES=0
OUT=#ADDFILE, delete
F1, "* No records selected - no data to ADD.", 1 left,
if ($ENTRIES=0)
H1, "ADD MEMO-FILE", 1 left
D1, ACCOUNT, 1 left
D2, SUFFIX, 1 left
D3, 0, 1 left
D4, %CYMD-TODAY, 1 left
D5, "The memo text", 1 left
D6, 0, 1 left
D7, 0, 1 left
TF1, "//", 1 left
END

XEQ #ADDFILE, QUIET
```

Discussion:

- You must specify *"LINES=0"* so that the header line is not repeated multiple times in the output file. With this setting, the output is considered one page of infinite length.
- Use the *"delete"* option on the OUT command to force the creation of a new file.
- Print the ADD command (ADD MEMO-FILE) on a header line. This line will be the first line in the output file.
- Use *"1 left"* for all of the output lines. This will cause the output to always start in the first column of the output file.
- Print the individual item values using D1, D2, D3 ... lines. This will print a set of values (one per line) for each of the selected records.
- Print "//" on a TF1 line. This will become the last line of the output file and will terminate the ADD command when the XEQ file is executed.

- Print a comment using an F1 line, when no records are selected (IF $ENTRIES=0). Using this technique, you can be sure that the XEQ file will always be created, even when no records qualify. When the file is XEQ'd, the comment will be displayed in the STDLIST.
- Finally, use the XEQ command to execute the ADD command using the file that you just created. The first line of the file will start the ADD command. Each of the item values will be provided by the contents of the XEQ file. The ADD will be terminated when the "//" line is read.
- The *QUIET* option will suppress the echo of the ADD command to the STDLIST. In general, it will be faster to add data in QUIET mode.

6.3 REPLACE Command [REPL]

Modify the value of *fields* for the entries in the current *select file*.

```
REPLACE field = "value"; [field = "value";] [...] END
```

Where:

field	is the name of a *field* to be modified by this command. Any *item*, *sub-item*, or *sub-field* may be modified, including *key items*.
value	is the new value for the *field*. The **same** value is used to update all records in the current *select file*. The value **must** be enclosed in quotes.

Notes:

This command can be used to modify any *set* located in a currently open *domain*. You must have write access to the *set* and to the *fields* that you wish to modify. All entries in the current *select file* will be updated. This command must be preceded by a FIND command. This command will be ignored if the Job Control Word PROTECT is equal to 1.

Example:

```
>FIND PROD-NO = 1006
>REPL STOCK-QT(2) = "100"; END
```

Null Values

If the `REPLACE` command appears in an *xeq-file*, you may use null values for one or more *fields*. Instead of specifying a value, use two double quotes (""). In this case, AskPlus will prompt for the value to be used in the REPLACE.

Example:

```
    REPL PRICE = ""; END
```

When this command is executed, AskPlus will prompt the user for the correct *value*.

```
What is the value of PRICE
```

The new *value* must be entered without quotes.

Limitation:

The REPLACE command has one significant limitation. The new value assigned to the field must be a constant. To assign a dynamically calculated value, use the

UPDATE statement (special form of the compute statement) in the REPORT command. See the next section for more details.

6.4 UPDATE Statement

A special form of the variable statement (REPORT and SAVE commands) is used to UPDATE a specified field with the value stored in the variable. This statement is much more powerful than the REPLACE command, because the value of the variable (and therefore the value used to update the field) can be calculated for each record. This statement can also be executed conditionally, which means that the update will only be executed when the specified condition is TRUE.

```
#variable, U[PDATE], field [, IF (condition)]
```

#variable	The name of the variable. The value of this variable will be used to update the value of field.
U[PDATE]	The UPDATE keyword signifies that this is a special version of the #variable statement. It can be abbreviated to "U".
field	The name of the field to be updated.
condition	If the IF parameter is specified, the UPDATE will only happen if the specified condition is TRUE.

This statement can be used to update values for any field in any *set* referenced in a current select file. You must have write access to the *set* and to the *fields* that you wish to update. All entries in the current *select file* will be updated, unless limited by the IF condition. This statement must be preceded by a FIND command. This statement will be ignored if the Job Control Word PROTECT is equal to 1.

UPDATE Example

This first example demonstrates how to update the credit limit based on the member's credit rating.

```
find primary=member-file
outsel=(loan-file)
(credit-rating ib 1, 99 and
 collateral = 35, 38, 40)

report
newreg increase:i2
newreg old-limit:i2
newreg new-limit:i2
#increase = 0
#increase = 99900, if (credit-rating ib 1, 19)
#increase = 66600, if (credit-rating ib 20, 39)
```

```
#increase = 33300, if (credit-rating ib 40, 99)
#old-limit=credit-lim
#new-limit=credit-lim + #increase
#new-limit, update, credit-lim, if (credit-lim > 0)
d1, account, tab1
d1, suffix, tab2
d1, #increase, tab3
d1, #old-limit, tab4
d1, #new-limit, tab5
end
```

In this example, loan-file records are selected based on the members credit-rating and the loan-file collateral code. In the report, three variables are created using the newreg statement. The first, increase, is used to calculate the amount to increase the credit limit. The second, old-limit, is used for audit purposes. The report prints the old limit as part of the output and therefore its value must be saved in a variable. The third, new-limit, contains the new credit limit. This variable is used in the update statement.

The next example demonstrates how to assign some random values to a field. This might be used in a test database, to populate a variety of values to a specific field. In this example we will select 500 members and then assign each member a credit rating of 4, 9, 14, ... up to 99.

```
find limit=500; member-file

report
newreg new-cr:i1=99
#new-cr = #new-cr + 5
#new-cr = 4, if (#new-cr > 99)
#new-cr, update, credit-rating
d1, account, tab1
d1, credit-rating, tab2
end
```

In this example, the variable new-cr is used to calculate a series of values to update the credit rating. The variable is initialized to 99. On each record, the variable is incremented by 5. If the value exceeds 99, the value is reset to 4. Using this logic, the first record will be updated with a 4. The subsequent records will be updated with 9, 14, 19, ... 94, 99. Once 99 is reached, the values will repeat starting at 4.

6.5 DELETE Command [DEL]

Delete **all** entries from **all** datasets recorded in the current *select file*. If the current select file contains entries from more than one dataset, this command will delete entries from multiple datasets. This command should be used carefully, deleted records **cannot** be automatically recovered. Beginning with version E.7.8.9, AskPlus will no longer automatically delete records from multiple datasets. If you want to delete records from multiple datasets, you must activate this feature using ALLOWMULTISETDELETE=1.

```
DELETE
```

This command must be preceded by a FIND command. In inter-active mode you will be asked for confirmation before the entries are actually deleted. In batch, the entries will be deleted without further confirmation. Make sure that your script is validated on a test database before using in production.

Always use the OUTSEL statement in the previous FIND command and specify which dataset(s) should be included in the select file. Do NOT use the OUTSEL=(@) form of the command. When using OUTSEL=(@), the contents of the select file can change if the selection criteria is changed. **This can have unexpected consequences and should be avoided.**

You must have write access to all *sets* described in the *select file* in order to delete the entries. If the *Job Control Word* `PROTECT` is equal to 1, this command will be ignored.

Example:

```
>FIND PRIMARY=setname
OUTSEL=(setname)
(Selection criteria)
Entries selected : xxx

>DELETE
xxx entries to be deleted. OK to proceed [no] ? YES
```

If the response is 'YES' all entries will be deleted.

DELETE Example

These examples demonstrate the impact that the current select file has on the entries that are deleted.

Example 1 - Delete ATM-FILE records based on status and expiration-date.

This is an example of BAD syntax - DO NOT USE.

```
FIND PRIMARY = ATM-FILE
OUTSEL = (@);
(ATM-FILE.MEMBER-STATUS <> 1 AND
 ATM-FILE.EXPIRATION-DATE <= %CYMD-TODAY )
```

```
Current OUTPUT select file is xxxx
using serial read
entries read 466 466
entries qualified 60
```

```
DELETE
```

```
Ready to delete entries from: ATM-FILE
entries qualified 4
entries to be deleted. OK to proceed [no] ? YES
```

This example successfully deletes records from the ATM-FILE, but the use of OUTSEL=(@) is dangerous. Watch what happens in the next example when the selection criteria is modified to include criteria involving the LOAN-FILE.

Example 2 - Delete ATM-FILE records based on status and expiration-date **and** LOAN-FILE criteria.

This is an example of BAD syntax - DO NOT USE.

```
FIND SEARCH
PRIMARY = ATM-FILE
OUTSEL = (@);
(ATM-FILE.MEMBER-STATUS <> 1 AND
 ATM-FILE.EXPIRATION-DATE <= %CYMD-TODAY AND
 LOAN-FILE.COLLATERAL = 111 AND
 LOAN-FILE.LAST-DATE <= %CYMD-TODAY )
```

```
Current OUTPUT select file is xxxx
using serial read
entries read 466 515
entries qualified 4
```

```
DELETE
```

```
entries to be deleted. OK to proceed [no] ?YES
```

This example will delete records from **both** ATM-FILE and LOAN-FILE. This is probably **not** what the user wants. The reason is the OUTSEL statement. The statement OUTSEL=(@) instructs AskPlus to record entries from ALL of the datasets mentioned in the selection criteria, in this case ATM-FILE and LOAN-FILE. See the next example, for the proper choice of OUTSEL.

Beginning with Version E.7.8.9, the delete would get the following message:

```
Ready to delete entries from: ATM-FILE, LOAN-FILE
entries qualified 4
Delete cancelled.  Cannot delete from more than one
dataset.
To allow multi-set deletes, set the system variable
ALLOWMULTISETDELETE=1
```

Example 3 - Delete ATM-FILE records based on status and expiration-date **and** LOAN-FILE criteria.

This is an example of GOOD syntax.

```
FIND SEARCH
PRIMARY = ATM-FILE
OUTSEL = (ATM-FILE);
(ATM-FILE.MEMBER-STATUS <> 1 AND
 ATM-FILE.EXPIRATION-DATE <= %CYMD-TODAY AND
 LOAN-FILE.COLLATERAL = 111 AND
 LOAN-FILE.LAST-DATE <= %CYMD-TODAY )

Current OUTPUT select file is xxxx
using serial read
entries read 466 515
entries qualified 4

DELETE

Ready to delete entries from: ATM-FILE
entries qualified 4
entries to be deleted. OK to proceed [no] ? YES
```

This example will delete records from **only** ATM-FILE. In this example, the OUTSEL statement instructs AskPlus to record entries from ATM-FILE only. Because the OUTSEL has been restricted to just ATM-FILE, the DELETE command will only delete the ATM-FILE records, it will NOT delete any records from LOAN-FILE.

Conclusion:

Always use OUTSEL=(setname) in the FIND command that precedes a DELETE command.

7 Utility Commands

7.1 ASSIGN Command [AS]

Assigns (sets) an AskPlus global option.

```
ASSIGN keyword [[=] options]
```

The ASSIGN command is used to change the default values of some AskPlus global options. Many of the options described in this command can also be set using commands in the Visimage/AskPlus configuration file (VISIMMGR). When appropriate, the reader will be referred to the *Visimage Administrator Manual* for a detailed discussion of the command. When the command is used in the configuration file, the keyword ASSIGN is not used. For example, the assign command "ASSIGN DATEFORMAT ..." will be the DATEFORMAT command in the configuration file.

Most of the assign commands consist of one or more keywords followed by a value. In some cases, the command syntax is more complex. The specific syntax of each of the commands will be discussed below in alphabetical order.

The following options may be set using the ASSIGN command:

BATCH	Terminate on error in batch mode.
DATEFORMAT	Default date format ($DATE)
FILTER	Record level security.
LANGUAGE	Default language (error messages)
LIMIT	Read and Select record limits.
LINK	Global link definitions.
LOCKOPTION	Default locking mode (LOCK/UNLOCK)
MACROFILE	Name of the current macro file.
PROGRAM	Default program name (EDITOR, SUPRTOOL, ...)
SCOPE	Specify the name of an Image dictionary.

Each of these keywords is described in more detail below.

BATCH

The batch command determines what happens in batch mode if AskPlus encounters an error.

```
BATCH [=] { ON | OFF }
   (Default: OFF)
```

With the default setting (BATCH = OFF), AskPlus will terminate with an error condition if an error occurs in batch mode. This command does not effect AskPlus when it is running interactively in session mode. Use BATCH = ON to force AskPlus to continue processing (in batch mode) after an error is encountered. Jobs that modify the database (add/update/delete) should always use the default setting (BATCH=OFF).

UNIX Considerations

- On UNIX, the system variable ABORT_BATCH_ON_ERROR must be set to "Y" for AskPlus to terminate in batch mode. In most cases, the script (ASKPLUS/askplus) has been configured to set this variable to "Y".
- On UNIX, the AskPlus run line option "-b" is used to indicate that AskPlus is running in batch. All AskPlus scripts should use the "-b" option.

Example

```
/ASKPLUS/askplus -b
...
>:echo $ABORT_BATCH_ON_ERROR
Y
>open badname;
Domain open error - non-existent file badname
Fatal error
End of Program
```

DATEFORMAT

The DATEFORMAT keyword determines the default data format that will be used by the system dates: $DATE, $DATE6 and $DATE8. This format is also used by the SORT statement when using the DATE option.

```
DATEFORMAT [=] { USA | EUROPE | INTERNATIONAL }
   (Default: USA)
```

Each setting has the following meaning:

USA	MM/DD/YY
EUROPE	DD/MM/YY
INTERNATIONAL	MM/DD/YY

The ASSIGN command changes the default date setting. The date may also be changed for the current report using the control statement (USA< EUROPE or INTERNATIONAL). See the *Visimage Administrator Manual* for more information on this command.

FILTER

The filter option applies record (row) level security to a particular dataset (or file). Records that meet the filter condition (condition is TRUE) cannot be selected by the current user.

```
FILTER [=] domain.set , condition ;
    (Default: none)
```

This command is most commonly set in a configuration file and not using the ASSIGN command. See the *Visimage Administrator Manual* for more information on this command.

LANGUAGE

Changes the default language used for displaying error messages and for responding to user input.

```
LANGUAGE [=] {ENGLISH | FRANCAIS | DEUTTSCH}

    (Default: English)
```

Askplus will use the specified language when displaying error messages. AskPlus will always recognize YES and NO as valid responses to prompts. The appropriate language equivalent will also be recognized when that language is chosen as the default.

LIMIT

Places limits on the number of records that can be read or selected in the FIND command.

```
LIMIT [=] option [=] value [ PERIOD [=] period [, period ] ]
   (Default: none)
```

This command is most commonly set in a configuration file and not using the ASSIGN command. See the *Visimage Administrator Manual* for more information on this command.

LINK

Defines a global link between two datasets or files.

```
LINK [linkname =] global_link ;
   (Default: none)
```

The LINK statement is available in the three main AskPlus commands (FIND, REPORT and SAVE). Links defined using the LINK statement or temporary and are valid only for the command in which they are used. The ASSIGN LINK allows a link to be defined globally for the current AskPlus session. This command is most commonly set in a configuration file and not using the ASSIGN command. See the *Visimage Administrator Manual* for more information on this command.

LOCKOPTION

Sets the default locking option to be used by the LOCK command.

```
LOCKOPTION [=] { BASE | SET }

   (Default: SET)
```

This option allows you to control the type of locking performed by the LOCK command. By default the lock will be made at the *set* level, however, choosing the value BASE, will cause the lock to be applied to the entire *base* containing the specified *set*. This option is only valid on HP 3000 (MPE) systems for Image databases.

Note: By default, AskPlus does not lock records. Therefore, it is not necessary to specify ASSIGN LOCKOPTION OFF. (In fact this option is not valid.)

MACROFILE

Specifies the name of the current macro file.

```
MACROFILE [=] file-name

   (Default:  KSMACRO)
```

This keyword is used to specify the name of the AskPlus macro file. The current macro file is used to save any new macros and to locate any existing macros. Only one macro file can be active at one time. See the MACRO command for information on creating macros. If no macro file has been specified, the file KSMACRO (in the current directory) will be used.

Example:

```
>newreg today:i2
>load today = %CYMD-TODAY
undefined macro CYMD-TODAY

>as macro /ASKPLUS/visimage/macro

>load today:i2 = %CYMD-TODAY
```

PROGRAM

Sets the default program (file) name for HP 3000 (MPE) programs.

```
PROGRAM progname [=] file[.group[.account]]
```

This keyword is used to supply a new name for some MPE programs executed by AskPlus. The *progname* can be one of the following:

progname	**Default value:**
EDITOR	EDITOR.PUB.SYS
QEDIT	QEDIT.PUB.ROBELLE
SUPRTOOL	SUPRTOOL.PUB.ROBELLE

Use "ASSIGN PROGRAM" to change these defaults. This keyword is only valid on HP 3000 (MPE) systems. These programs are actually started by the EDITOR, QEDIT and SUPRTOOL commands. See also the ":system" command which will run and command/program on any supported platform.

Examples

```
>AS PROGRAM EDITOR = QUAD.UTIL.SYS
>AS SUPRTOOL = SUPRTOOL.UTIL.SYS
```

SCOPE

Opens the specified dictionary or resets the scope to use only IMAGE item names.

```
SCOPE [=] {dictname | IMAGE}
   (Default: IMAGE)
```

This command is most commonly set in a configuration file and not using the ASSIGN command. See the *Visimage Administrator Manual* for more information on this command.

7.2 COMMENT Command [CO]

Define a comment line.

```
COMMENT text
```

The AskPlus language provides three different ways to specify comments. Including comments in your code is a highly recommended practice. In batch, comments entered in the job/script will normally be echoed in the STDLIST. Comments from XEQ files may or may not appear depending on how the comment was specified and whether or not the QUIET option of the XEQ command was used.

COMMENT Command

The comment command is valid only at the command level of AskPlus. The keyword (COMMENT) may be followed by any text except a semi-colon. Since the semi-colon is considered by AskPlus to be a command terminator, any text that follows the semi-colon will be treated as a new command and not part of the comment.

Comments from an XEQ file entered using the comment command will **not** appear on the STDLIST if the XEQ file is executed with the QUIET option.

Example

```
>Comment Select customers whose first name is JOHN.
>Find cust-name = "JOHN @"(smart)
```

* Command / Statement

A comment may also be entered using the '*' command or statement. This type of comment is similar to the comment command but has two important differences:

- The '*' comment is valid both at the command level and also at the statement level. Comments entered using a '*' may be included as statements inside a FIND< REPORT or SAVE command.
- Comments from an XEQ file entered using the '*' command will always be included in the STDLIST, even when the QUIET option is specified.

Example

```
>Find primary=customer-master
>>* Select customers whose first name is JOHN.
```

```
   >>cust-name = "JOHN @"(smart)
```

<< comment >>

The final method of entering comments is to include the comment between pairs of angle brackets << ... >>. This type of comment may be entered at any point (except inside a quoted string), even in the middle of another command or statement. This type of comment is the most flexible and is often used to temporarily comment out sections of code that should not be executed.

Example

```
   >Find primary=customer-master
   >>* Select customers whose first name is JOHN.
   >>cust-name = <<"JOHN @">>"J@"(smart)
```

7.3 DB Locking Commands

The locking commands LOCK and UNLOCK are used to LOCK or UNLOCK specific datasets. These commands are supported only for (Turbo-)Image and Eloquence databases.

LOCK Command [LOC]

Locks a dataset or list of datasets.

```
LOCK  dataset [,dataset] [...]
```

Where :

dataset	is the name of a dataset to be locked.

This command locks the specified datasets which will remain locked until they are released. This command is only available in an XEQ file or in batch mode. This restriction ensures that a dataset will not be left locked by error. All sets will be automatically released at the end of the XEQ file. A set can be explicitly released using the UNLOCK command, or by closing the domain.

If you attempt to lock a *set* that is currently locked by another user, and the session is interactive, you will be asked if you wish to wait.

```
Data set locked by another process
Do you wish to wait [no] ?
```

If the reply is 'YES', AskPlus will suspend until the *set* is released by the other process and your lock request can be successfully executed. If the reply is 'NO', the command is ignored. In batch mode, AskPlus will always suspend.

Example

```
>XEQ XLOCK
LOCK ORDERS, PRODUCTS, CUSTOMERS

FIND ALL ORDERS
Entries selected : 17

XEQ RTOTAL
```

```
    ...
   end of Xeq file         << UNLOCK ALL implicit >>
```

UNLOCK Command [UNL]

Unlocks (releases a lock) for one or more datasets previously locked by the LOCK command.

```
UNLOCK    {dataset [,dataset] [...]}
          {database                }
          {ALL                     }
```

Where:

dataset	Unlocks the list of specified datasets.
database	Unlocks all sets currently locked in the specified database.
ALL	Unlocks all sets currently locked.

This command is only available in batch mode or in an XEQ file. An '`UNLOCK ALL`' is automatically executed at the end of an XEQ file, before returning to interactive mode.

Example

```
>XEQ XUNLOCK

LOCK ORDERS, PRODUCTS, CUSTOMERS

FIND ALL ORDERS
Entries selected : 17

XEQ RTOTAL

 ...

UNLOCK BASEA

FIND ALL ENTRIES
Entries selected : 29

XEQ RSUPP

End of Xeq file
```

7.4 DB Logging Commands

The logging commands DBBEGIN, DBEND and DBMEMO update the database log file when logging has been enabled for one of the open databases. These commands are supported only for (Turbo-)Image databases on HP 3000 (MPE) systems. These commands are only valid if the database is enabled for logging.

The commands are:

```
DBB[EGIN] database[, comment]
DBE[ND]   database[, comment]
DBM[EMO]  database[, comment]
```

Where:

database	is the name of an open Image database.
comment	is text to be written to the *logfile*.

DBBEGIN

Writes the specified comment to the logfile indicating the start of a logical transaction. A call to DBBEGIN, must be matched by a corresponding call to DBEND.

DBEND

Writes the specified comment to the logfile indicating the end of a logical transaction. A call to DBEND, must be preceded by a call to DBBEGIN.

DBMEMO

Writes the specified comment to the logfile.

These commands are equivalent to the TurboImage intrinsics of the same name. For more information on these commands consult the Image Reference Manual.

Example:

```
>DBBEGIN BASEA,"Transaction to update Price."
>DBMEMO BASEA, "Updating product 1001"
>FIND PRODUCTS.PROD-NO = 1001
>REPL SELL-PRICE = "600"; END
>DBMEMO BASEA, "Updating product 1002"
>FIND PRODUCTS.PROD-NO = 1002
```

```
>REPL SELL-PRICE = "2100"; END
>DBEND BASEA, "Prices successfully updated."
```

7.5 DISPLAY Command [D]

Display the contents of a file on the standard list device.

```
DISPLAY filename
```

This command displays the contents of the specified file on the standard list device. In interactive mode this is the terminal. The system commands print (MPE) and cat or more (UNIX) perform a similar function. However, you must prefix the system commands with a colon.

Example:

```
>DISPLAY XREP.TEST

REPORT LP
S1, CUST-NO
 ...
END
```

7.6 EXIT Command [E]

Terminate or suspend AskPlus execution.

```
EXIT [KILL]

   (Default: suspend)
```

This command will stop execution of the program. If the keyword KILL is specified, AskPlus will terminate on exit not allowing a later re-activation. See below for more details. The KILL parameter is syntactically valid on UNIX systems, but serves no special purpose.

KILL Parameter

When exiting the program on MPE systems, AskPlus will determine if it (AskPlus) was started from the command prompt or from another program that activated AskPlus as a son process. If AskPlus was started from the command prompt, AskPlus will close all the opened domains and then exit. If, however, AskPlus was started from a program capable of *reactivating* AskPlus (such as `QEDIT` or `MPEX`), AskPlus will only suspend (rather than terminate) so that the father program can reactivate it later. In this case, if you re-activate AskPlus you will be at the (*prompt* '>') level and your original environment will be unchanged (bases opened, select file active, etc.). Use the "KILL" parameter to force AskPlus to terminate on exit, rather than suspend.

Note:

Programs such as `QEDIT` and `SUPRTOOL` are also held when executed by AskPlus. These programs are then re-activated as needed.

7.7 Global Variables

AskPlus variables (also called registers) can be created as global variables or as local variables. Global variables retain their value from creation until they are deleted (purged) or until AskPlus exits. Global variables are created using the NEWREG **command**. In contrast, local variables retain their value only for the command in which they are created. Local variables are created using the NEWREG **statement** in either the FIND, REPORT or SAVE commands.

Global variables are useful for constant value (like dates) that will be used in more than one command, for example in both a FIND (as a selection criteria) and in the REPORT (as information printed on the page header). They are also useful, in some instances, for passing a result from one report to another. For example, report #1 might calculate a total balance or count. This total can be used in a subsequent report to calculate percentages. In general, it is recommended that you use local variables unless you have a specific reason to make the variable global.

AskPlus provides three commands for working with global variables:

NEWREG	This command is used to create a new variable.
LOAD	This command is used to load the variable with a new value.
PURGEREG	This command is used to purge (delete) the variable.

This commands are discussed in more detail below.

NEWREG Command [N]

Creates a global register.

```
NEWREG [#]variable:[scale]typelen [[=] expr]
```

Where:

#variable	is the name of the variable. Since this command only works with variables, the '#' prefix is not required..
scale	is the number of occurrences of the variable. The default scale is 1 and does not need to be specified. A scale larger than one indicates that the variable is an array.
typelen	Is the type and length of the variable. See Appendix B for a list of valid types and lengths.
expr	Is the initial value of the variable. If no initial value is specified, numeric variables will be initialized to zero and character variables

will be initialized to blank. If the variable is an array (has a scale greater than 1), the initial value will be applied to all elements of the array. The special operator "!index(@)" is allowed when loading the initial value of array variables.

The syntax of the NEWREG command is identical to the syntax of the NEWREG statement. (See ***Chapter 4, Reporting Data*** (Page 81) NEWREG statement for more information.)

Discussion:

- Variables created at the command level are global variables..
- The initial value of the variable can be set as part of the NEWREG command.
- The value of the variable can be changed using the LOAD command (see below).
- The value can be set using a constant or an expression. The expression **may not** include any **item** names, but may include other variables and macros.
- The variable can be deleted (purged) using the PURGEREG command (see below).
- The current values of all global variables can be displayed using the "SHOW REGS" command.

Examples:

```
>newreg start-date:x8=?Start_Date_CYMD
>newreg end-date:x8=?End_Date_CYMD
>newreg num-days:i1=%CYMD-DIFF(#start-date, #end-date)
>sh regs
Registers   = start-date:1X8 = 20060501
              end-date:1X8 = 20060530
              num-days:1I1 = 30
```

LOAD Command [LO]

Loads (changes) the value of a global variable.

`LOAD [#]`*variable*[(*index*)] [=] *expr* [, IF *cond*]

Where:

#variable	Is the name of a variable that was created by a previous NEWREG command. The '#' is optional in the variable name.
index	If specified, is the index (or subscript) of an array variable. The value of the specified index is changed by this command. An index of '@' may be specified, in which case all of the array values will be loaded.
expr	Is the expression that is evaluated to update (load) the variable.
cond	If specified, the value of the variable is only changed when the expression evaluates to TRUE.

Discussion:

The LOAD command is functionally equivalent to the '#variable' compute statement in the FIND, REPORT and SAVE commands. There are however, some differences:

- The LOAD command is executed once (and the expression is evaluated once). The '#variable' compute statement is evaluated for every record.
- The expression for the LOAD command must not include any item names.

Examples:

```
>LOAD START-DATE = %FIRST-DAY-PRI-MO
>LOAD END-DATE = %LAST-DAY-PRI-MO
>LOAD DATES(@) = (#START-DATE + !index(@) - 1)
```

PURGEREG Command [P]

Deletes (purges) a global variable (register).

```
PURGEREG  {[#]variable}
          { @      }
```

Where :

#variable	is the name of the global variable to be purged. The '#' is optional in the variable name.
@	indicates that ALL global variables should be purged.

Discussion:

The PURGEREG command purges (deletes) an existing global variable. Global variables are automatically deleted on exit, so there is no need to purge a variable unless you want to re-create it with a different type or length (typelen). A NEWREG statement that tries to re-create an existing global variable will fail if the variable is not purged first. The value of a global variable can be changed using the LOAD command.

Examples:

```
>PURGEREG START-DATE
>PURGEREG END-DATE
```

7.8 LIST Command [LI]

Prints a quick list of entries from a data set or file with automatic formatting and headings.

```
LIST [WIDTH=nn] {set-name } [FOR condition] [END]
                 {item-list}
```

The LIST command displays records for the specified dataset (or item list) to the terminal, or to the location specified by the most recent OUT command. If a FOR condition is specified, only records that meet the condition are listed. The number of columns displayed is limited by the output width. This command only operates on a single dataset or file.

Where:

nn	When specified, is the character width of the output.
set-name	Is the name of a data set or file in an open domain.
item-list	Is a list of data items separated by commas. All data items must belong to the same dataset or file.
condition	When specified, is a condition (criteria) which must be true for the record to be displayed.

Example

List memo-file records (NUMBER=6001) to a file called MEMOLIST.

```
>OUT = MEMOLIST
>LIST WIDTH=200 MEMO-FILE FOR NUMBER = 6001
>OUT=TERM
```

List three items from a dataset to the terminal.

```
>LIST SHARE-FILE.ACCOUNT, SUFFIX, BALANCE &
>>FOR SUFFIX = 0 AND BALANCE < 2500
```

7.9 MACRO Command [MA]

Creates a *macro* and saves the definition in the current macro file.

```
MACRO name [(parms)] = definition
```

Where:

name	Is the name of the macro being defined.
parms	Is a list of parameters that will be substituted into the macro each time the macro is used. Parameters are not required, however, if the macro definition contains parameters, then they must be specified each time the macro is used.
definition	Is the macro definition. This definition will replace the macro name, each time the macro is used.

Discussion:

Macros are definitions of names or formulas which can then be used anywhere in an AskPlus command or statement. When used, the macro name (and optional parameters) is replaced by the macro definition. A macro can be used anywhere provided the definition makes sense in the context where it is used.

Refer to the *Visimage Administrator Manual* for a detailed discussion of creating and using macros.

Examples:

```
>macro MOD(nbr, div) = (nbr - ((nbr) / div * div))
>macro MOD10(nbr) = %MOD(nbr, 10)
>macro ACCT-SFX(acct,sfx) = !FORMAT("999999999", acct)
&
 splice "-" splice !FORMAT("99", sfx)
```

Note: If a macro definition extends for more than one line, you must enter the '&' continuation character to continue the definition on subsequent lines.

7.10 MAKETEMP Command [MAKET]

Loads the specified global variable with a unique (temporary) file name.

```
MAKETEMP #variable
```

Where:

#variable	Is the name of the global variable to be loaded with a unique filename.

Discussion:

This command is used to load a global variable with a unique filename. AskPlus creates the unique filename at run time when the command is executed. This guarantees that if the same job is run simultaneously, AskPlus will use a unique filename for each run.

A filename loaded in a global variable may be used in the INSEL/OUTSEL statements, the OUT statement or the XEQ command. See the OUTSEL statement (*Chapter 3, Data Selection*) for an example. Also see the XEQ command later in this chapter for another example.

It is recommended that the variable be created as an X100 to ensure that a fully qualified filename will fit.

Example:

```
>NEWREG FNAME:X100

>MAKETEMP FNAME

>SH REGS
Registers   = fname:1X100 = ./RG33200
```

7.11 OUTPUT [OUT]

This command/statement determines where AskPlus output will be sent. If entered as a command (prior to a REPORT), the OUTPUT will be re-directed as specified until the next OUTPUT command. If entered as a statement in the REPORT or SAVE commands, the output will be redirected only for the current command and then will revert back to display (TERM).

```
OUTPUT [=]     { TERM                          }
               { filename [, file_parms]   }
               { LP [, lp_parms]           }
               { $devname   [, lp_parms] }
               { #variable [, parms]           }
(Default: TERM)
```

Where:

TERM	By default, output is displayed on the user's terminal. On the HP3000, the TERM designation is associated with the symbolic file name ASKOUT and is the default equivalent of $STDLIST.
filename	AskPlus output is redirected to a file with the specified name.
LP	By default, represents the system's printer. On the HP3000 this is associated with the symbolic file name `ASKLIST` and is the default equivalent of the LP class.
$*devname*	On MPE, this option directs output to the specified device name or class. The setting $LP is equivalent to LP. On UNIX, this option redirects output to the printer with the specified devname.
#*variable*	Like the previous two options, this choice redirects output to a file or a device, depending on the value of the variable. AskPlus uses the current value of the variable to determine the output destination. The value of the variable can be a file name, or a $devname.

File Parameters:

File parameters are entered after the filename and separated by a comma. Multiple options may be specified, each one separated by a comma.

DISC=*nnn*	On HP3000 systems, this option determines the maximum number of records (*nnn*) in the output file. On UNIX, this parameter is ignored.
REC=*ll*	This option specifies the maximum length (*ll*) in characters of each record. See also the WIDTH statement in the REPORT Command.
FILL	On UNIX systems, this option causes each output record to be padded (filled) with blanks. The record will be padded up to the specified width or record length. MPE files are always fixed length with padded blanks. UNIX files are variable length and, by default, are not padded with blanks.
APPEND	If the file already exists, new records are appended to the end of the file.
DELETE	If the file already exists, this option will purge the old file and create a new file without being prompted for confirmation.

LP Parameters:

LP parameters are entered after the device name (or LP) and separated by a comma. Multiple options may be specified, each one separated by a comma.

message	On MPE, the specified message will be sent to the master console (operator message) in order to prompt the operator to install a special type of paper (printed forms or labels, for example).
COPIES=*nn*	Specifies the number of copies to be printed of the same report. Default value is 1.
OUTPRI=*nn*	Specifies listing output priority (1 to 13). The default value is 8. MPE only.

Discussion:

- The `OUTPUT` **command** re-directs output from the REPORT and FORM commands as well as from some utility commands.
- All error messages, warnings and prompts are sent to `ASKOUT` and are not redirected by the `OUTPUT` command.
- An output assignment remains in effect until the current command is terminated, or in the case of the OUTPUT **command**, a new OUTPUT **command** is entered.

Examples:

```
>out = lp
>form
>out = term

>out $lp1
>report
  ...
>>end
```

Example on MPE using a file equation:

```
:FILE ASKLIST=FILE1;DEV=DISC;LIMIT=100000
:ASKPLUS PUBTCP
>out = lp
...
```

Example on UNIX using system variables:

```
export ASKOUT=FILE1
/ASKPLUS/askplus
>newreg #askout:x50=^ASKOUT
>out = #askout, delete
...
```

7.12 System Command

Most system commands and programs can be executed by AskPlus. Any command prefixed by a colon (:) will be passed directly to the host operating system for execution. It will not be parsed by AskPlus.

```
:command
```

Discussion:

AskPlus does not recognize the command as a system command unless you prefix the command with a colon (:). AskPlus uses a colon to signify a system command on all operating systems. AskPlus will pause until the command finishes processing.

MPE Examples

```
>:listf @01.DATA
```

```
>:listspf
```

UNIX Examples

```
>:rm SDFILE01
```

```
>:vi
```

Other MPE Program Commands

AskPlus also provides several different commands that launch specific MPE programs.

EDITOR Command [ED]

Invoke the HP 3000 (MPE) standard text editor.

```
EDITOR
```

This command executes the MPE standard text editor (EDITOR.PUB.SYS). A different editor can be called by using the ASSIGN PROGRAM command. To invoke an editor on UNIX, enter the appropriate system command.

QEDIT Command [Q]

Invoke the QEDIT(Tm) text editor on HP 3000 (MPE) systems.

```
QEDIT
```

This command is used to start the ROBELLE text editor (QEDIT.PUB.ROBELLE) if it is available on your system. Use the ASSIGN PROGRAM command to force AskPlus to launch a different copy of QEDIT.

SUPRTOOL Command [SUPR]

Invoke the utility program `SUPRTOOL`(Tm).

```
SUPRTOOL
```

Runs the program SUPRTOOL.PUB.ROBELLE, a general-purpose utility for accessing and copying datasets and files.. Use the ASSIGN PROGRAM command to launch a different copy of SUPRTOOL.

7.13 SHOW Command [SH]

Display the current values for a variety of AskPlus environment settings.

```
SHOW [keyword]
   (Default: ALL)
```

The SHOW command displays information back to the STDLIST about the current environment: open domains, dictionary scope, select file, priority set list, global variables and syntax. By default, all of this information will be displayed. To see only one item, specify the appropriate keyword.

Where:

Keyword is one of the following:

Keyword:	**Meaning**
FILES	Display all open domains (databases and files.)
REGISTERS	Display all global variables.
SCOPE	Display the scope (dictionary) of each open domain.
SELECTFILE	Display information about the current select file.
SETS	Display the list of priority sets.
SYNTAX	Display the current syntax (ASKD or ASKC)
ALL	Display ALL of the above information.

Example:

```
>sh all

Select-File = ./ASK24300
    Primary = MEMBER-FILE
              SHARE-FILE
    Entries = 924

Syntax      = ASKD

Scope       = MEMBRS   No dictionary
```

```
Domain      = membrs     MEMBRS    IMAGE
Sets        = MEMBER-FILE
              SHARE-FILE
Registers   = start-date:1I2 = 20060101
              end-date:1I2 = 20061231
```

Discussion:

Each of the keywords is discussed in more detail below.

FILES:

This option displays a list of the currently open domains (databases and files). For a detailed discussion of opening and closing domains refer to *Chapter 2, Accessing Data*.

REGISTERS:

This option displays a list of all global variables that are currently defined. Refer to *Global Variables*, earlier in this chapter, for information on creating and using global variables.

SCOPE:

This option displays the current scope for each of the open domains. The scope is either the name of a dictionary (HP 3000 only) or simply IMAGE. Dictionaries and the SCOPE command are discussed in more detail in the *Visimage Administrator Manual*.

SELECTFILE:

This option displays information about the current select file. It displays the name of the select file, the list of sets saved in the select file and the current number of entries stored in the select file. Refer to *Chapter 3, Data Selection* for more information on selecting data and the importance of select files.

SETS:

This option displays the current list of priority sets. Priority sets are discussed in more detail in *Chapter 3, Data Selection.*

SYNTAX:

This option displays the current syntax mode which in most cases will be ASKD. It is possible to force AskPlus to accept ASK Version C syntax, but that is only done when compatibility is required with legacy ASK C code.

7.14 SELECT-FILE Command [SEL]

Specify the name of a file to be used as the current select file.

```
SELECT-FILE [=] filename
```

The select file is where AskPlus stores the results of a FIND command. All records that qualify for the selection are stored in the select file where they can be accessed by a subsequent REPORT or SAVE command. The INSEL and OUTSEL statements are the most common way of specifying the name (and contents) of the select file. Consequently, the SELECT-FILE command is not used very often. It is recommended that these statements be used instead of the SELECT-FILE command.

The name of the current select file may be displayed using the "SHOW SELECTFILE" command.

7.15 VERSION Command [V]

Display the current AskPlus version number.

```
VERSION
```

This command will display the current version of AskPlus.

7.16 VINFO Command

The VINFO command provides information similar to the FORM command. The VINFO command is actually an internal command used by Visimage to retrieve information about the current environment and to build the local dictionary. Some of the VINFO parameters are described here as they may be useful.

```
VINFO DOMAIN, domain
VINFO SET, [domain.]set
VINFO SET PATH, [domain.]set
VINFO ITEM, [set.]item
```

Where:

DOMAIN	Displays a list of all of the sets in the specified domain. No item information is included.
SET	Displays a list of all of the items in the specified set.
SET PATH	Displays a list of all of the accessible secondary sets for the specified set. This command is used to populate the Visimage secondary topics list.
ITEM	Displays a list of all the sets that contain the specified item.
domain	Is the name of a currently open domain.
set	Is the name of a set (or file) in a currently open domain.
item	Is the name of an item in a currently open domain.

Examples:

```
VINFO DOMAIN, MEMBRS

VINFO SET, MEMBER-FILE

VINFO SET PATH, MEMBER-FILE

VINFO ITEM, ACCOUNT
```

7.17 XEQ Command [X]

Execute a file containing AskPlus commands.

```
XEQ {filename } [,IF cond] [,Q[UIET]] [,NODATA]
    {#variable}
```

Where:

filename	Is the name of the file containing the AskPlus commands to be executed. On HP 3000 (MPE) systems, the name must be specified using "file.group.account" syntax. On other operating systems, the file name must be specified using the syntax appropriate for those systems.
#variable	Is the name of a variable that contains a valid filename. Instead of specifying the name of the file in the XEQ command, specify the name of a global variable that **contains** the file name.
cond	If specified, is the name of a condition that determines whether or not the XEQ files should be executed. If the condition evaluates to TRUE, the file is executed, otherwise a message is printed that indicates the file was not executed.
QUIET	By default, the commands in the XEQ file will be displayed to the STDLIST during execution. If the QUIET keyword is specified, the commands are not displayed, however, the results of the commands are displayed.
NODATA	This keyword tells AskPlus to prompt the terminal, and not the XEQ file when requesting additional information. This keyword is not normally needed, since for most prompts, AskPlus will default to prompting the user and not the XEQ file.

Discussion:

The XEQ command executes the commands contained in the specified file. Using XEQ files is useful in a variety of situations. XEQ files are often used to execute code that may be used in more than one job or process. XEQ files can also be used by using AskPlus to create the XEQ file from the data that is processed.

The XEQ file must be an ASCII file and can be created by any text editor. XEQ files can also be created as output using the AskPlus REPORT command. A good example of this using the ADD command is repeated below. Notice in the example, the file name is stored in a global variable which was loaded using the MAKETEMP command. Also notice, that the QUIET option is specified, so that the actual data is not echoed back to the STDLIST.

ADD Example using XEQ

A common method of adding data, is to use AskPlus to create an XEQ file which contains both the ADD command and all of the data values that will be used to add the data. This is done using the REPORT command to create the XEQ file.

In the following example, dataset records are selected based on the value of a specific flag. The selected records are then used to populate the values that will be used to add the data. In this example, we will supply all of the values and will not use the "USING filename" syntax.

```
NEWREG ADDFILE:X100

MAKETEMP #ADDFILE

FIND SHARE-FILE.CONTROL-FLAGS(21) = 1

REPORT
LINES=0
OUT=#ADDFILE, delete
F1, "* No records selected - no data to ADD.", 1 left,
if ($ENTRIES=0)
H1, "ADD MEMO-FILE", 1 left
D1, ACCOUNT, 1 left
D2, SUFFIX, 1 left
D3, 0, 1 left
D4, %CYMD-TODAY, 1 left
D5, "The memo text", 1 left
D6, 0, 1 left
D7, 0, 1 left
TF1, "//", 1 left
END

XEQ #ADDFILE, QUIET
```

Discussion:

- You must specify *"LINES=0"* so that the header line is not repeated multiple times in the output file. With this setting, the output is considered one page of infinite length.
- Use the *"delete"* option on the OUT command to force the creation of a new file.

- Print the ADD command (ADD MEMO-FILE) on a header line. This line will be the first line in the output file.
- Use *"1 left"* for all of the output lines. This will cause the output to always start in the first column of the output file.
- Print the individual item values using D1, D2, D3 ... lines. This will print a set of values (one per line) for each of the selected records.
- Print "//" on a TF1 line. This will become the last line of the output file and will terminate the ADD command when the XEQ file is executed.
- Print a comment using an F1 line, when no records are selected (IF $ENTRIES=0). Using this technique, you can be sure that the XEQ file will always be created, even when no records qualify. When the file is XEQ'd, the comment will be displayed in the STDLIST.
- Finally, use the XEQ command to execute the ADD command using the file that you just created. The first line of the file will start the ADD command. Each of the item values will be provided by the contents of the XEQ file. The ADD will be terminated when the "//" line is read.
- The *QUIET* option will suppress the echo of the ADD command to the STDLIST. In general, it will be faster to add data in QUIET mode.

8 Expressions

8.1 Overview

This chapter discusses the syntax of expressions and provides more details on how expressions can be used in AskPlus. The available special operators and system variables are documented in detail.

8.2 Expressions

Perhaps the most common element in AskPlus is the idea of an expression. An expression is a calculation involving a combination of different objects (items, sub-items, macros, constants, ...) called operands, connected by symbols called operators which describes an operation between the objects. The calculation QTY * PRICE is an example of an expression. In fact, in its simplest form, an expression can be a single item like ORDER-DATE.

In AskPlus, expressions can be used to search for data in a database (selection criteria), to define a column in a list or a field in a report, to define conditions for printing or to perform a variety of other actions. In AskPlus, it is almost always possible to use an expression in place of a simple object.

As defined above, an expression is a combination of operands and operators. In the expression, QTY * PRICE; QTY and PRICE are both operands and the symbol * is an operator. Let’s look at each of these in more detail:

When writing an expression, always separate the operands from the operators with a space.

Types of expressions

The result of an expression can either be arithmetic, logical, or character.

Arithmetic expressions

The result of an arithmetic expression is always numeric. Relational operators such as > (greater than), < (less than), etc.., and logical operators such as AND and OR are not used in arithmetic expressions. The following example is an arithmetic expression:

```
(SELL-PRICE - BUY-PRICE) * QUANTITY
```

Logical expressions

The result of a logical expression is always TRUE or FALSE. Logical expressions use all the operators. The following example is a logical expression:

```
PRODUCT-NO IB 1000, 1999 AND
(SELL-PRICE - BUY-PRICE) * QUANTITY > 1000
```

Character Expressions

The result of a character expression is always character. Character expressions can use the string operators CAT, JOIN and SPLICE. For example:

```
FIRST-NAME JOIN LAST-NAME
```

Operands

The most common operand in AskPlus is an item from the database. However, there are many more types of operands. An operand is usually one of the following:

- Item
- Sub-item
- Constant
- Wildcard Value
- User Prompt
- Variable
- System variable
- Special Operator
- Macro

In general, any operand can be combined with any other operand provided they are joined by a valid operator and the result of the expression makes sense. Types of expressions and the different operands are discussed below.

Operators

Operators perform operations on operands. The most common operators are the four basic ones used in arithmetic: add, subtract, multiply and divide. AskPlus also has relational operators for logical expressions, conditional operators and string operators.

Arithmetic Operators

AskPlus uses the standard symbols as arithmetic operators:

+	Add

-	Subtract
*	Multiply
/	Divide

In addition to these, there is an additional divide operator called DIVR. By default the / operator will **truncate** the result when performing integer division, while DIVR will always **round** the result. If the **Rounding** option in the **List** or **Report Option** windows is set, these operators are identical. For real number operands, / and DIVR will always produce the same result because the decimal part is kept in the result. There is no truncation or rounding.

The following example shows the difference between truncation and rounding:

```
5 / 3 = 1

5 DIVR 3 = 2

5 DIVR 4 = 1
```

Relational Operators

Relational operators are used to establish logical relationships (comparisons) between operands. The result is called boolean (or logical) and is always TRUE or FALSE. AskPlus allows relational operators to be specified with symbols or with a two-letter combination.

=	EQ	Equals
<>	NE	Not equals
<	LT	Less than
>	GT	Greater than
<=	LE	Less than or equal to
>=	GE	Greater than or equal to
IB	IB	Includes between

Let’s consider some of these in more detail.

Equals and Not Equals

In the simplest case, these operators compare two operands and decide whether or not they are equal or not equal. For convenience, it is possible to compare an operand to a list of operands separated by commas. This shorthand corresponds to an abbreviation of a more complex condition constructed with logical ORs (in the case of equality), or logical ANDs (in the case of inequality). For example:

```
PROD-NO = 1000, 1005, 1010
```

Is equivalent to and much simpler than:

```
PROD-NO = 1000 OR
PROD-NO = 1005 OR
PROD-NO = 1010
```

Similarly:

```
PROD-NO <> 1000, 1005, 1010
```

Is equivalent to:

```
PROD-NO <> 1000 AND
PROD-NO <> 1005 AND
PROD-NO <> 1010
```

IB operator

The IB operator is a relational operator that has a special format because it requires two operands separated by a comma. This operator is the same as using >= and <= joined with an AND.

For example:

```
ITEM IB "value1","value2"
```

Is equivalent to:

```
(ITEM1 >= "value1" AND ITEM1 <= “value2”)
```

Conditional Operators

Conditional operators are used to join two logical operands. Conditional operators, like relational operators, always return a result of TRUE or FALSE.

AND	TRUE if both TRUE
OR	TRUE if one or both is TRUE
XOR	TRUE if only one is TRUE
NOT	TRUE if FALSE, FALSE if TRUE

AND

The result of joining two relational operands with an AND will be TRUE, only if both of the operands are TRUE. If either of the operands is FALSE, the result of joining them with an AND will also be FALSE.

OR and XOR

The result of joining two relational operands with an OR will be TRUE, if one or both of the operands is TRUE. If both of the operands are FALSE, the result of joining them with an OR will also be FALSE. The operator XOR stands for exclusive OR.

Using this operator, the result will be TRUE only if one, **but not both**, of the operands is TRUE. If both operands are TRUE, or both are FALSE, the result of XOR will be FALSE.

NOT

The operator NOT is used with a single operand to reverse the value of the operand. Using NOT will change a TRUE value to FALSE and a FALSE value to TRUE. NOT is useful in a selection criteria to exclude specific values. Enter a condition that selects the values you want to exclude, then enclose the condition in parentheses and prefix it with NOT. All records that match the condition in the parentheses will NOT be selected.

String Operators

String operators are used to concatenate character operands together. There are three string operators (CAT, JOIN and SPLICE). If these operators are used with numeric operands, the operands will be automatically converted to character before the string operation is performed. The difference between these operators is the number of spaces that is left in the result between the two operands.

CAT	Keep all blanks
JOIN	Leave one blank between
SPLICE	Leave no blanks.

CAT is used to join two objects end-to-end without changing their length. JOIN reduces the number of blanks separating the end of the first operand and the beginning of the second operand to one, while SPLICE suppresses all the spaces between the two concatenated character strings. For example:

```
" AB " CAT " CD "    produces " AB  CD "
" AB " JOIN " CD "   produces " AB CD "
" AB " SPLICE " CD "       produces " ABCD "
```

A common use of JOIN and SPLICE is formatting the last line of an address:

```
CITY SPLICE “,” JOIN STATE JOIN ZIPCODE
```

To force the result of this expression to have a fixed length we can add the !CVT special operator to convert to a character field, length 30, which will pad the expression with the right number of trailing blanks.

```
!CVT:X30(CITY SPLICE “,” JOIN STATE JOIN ZIPCODE)
```

Binary Concatenation

Another concatenation operator, BCAT, is used to concatenate two operands of any type without converting to character. The result of this operation usually produces a result that has no specific type and that cannot be printed. This operator is usually reserved for use in a LINK statement. It is used in cases where the key value is a concatenation of two fields. No character conversion is performed with BCAT.

Priority of operators

When AskPlus evaluates expressions, it uses the following rules to determine priorities. All expressions enclosed in parentheses are evaluated first. To determine priority when there are no parentheses, the following table is used.

Arithmetic Operators:

Operator	**Meaning**	**Priority**
* / DIVR	Multiply Divide Divide with Rounding	1
+ -	Add Subtract	2

Value Separators:

Operator	**Meaning**	**Priority**
,	To separate a list of values. Use with =, <> and IB.	3

Relational Operators:

Operator	**Meaning**	**Priority**

= or EQ <> or NE IB > or GT < or LT >= or GE <= or LE	Equals Not Equals Includes between value1, value2 Greater Than Less Than Greater than or equal to Less than or equal to	4

Conditional Operators:

Operator	**Meaning**	**Priority**
AND	Both conditions must be true.	5
OR	One of the conditions must be true	6

8.3 Items and Sub-items

An item is a field that exists in a dataset (or file). In its simplest form, an item is a single name like ACCOUNT, DATE or PROVIDER. To eliminate possible confusion with other items that share the same name, an item may be fully qualified with the name of the dataset, the name of the database or both. Use a period to separate the database, dataset and item names.

All of the following are valid item names.

```
INVOICE-DATE
INVOICE-HEADER.INVOICE-DATE
FINDB.INVOICE-HEADER.INVOICE-DATE
```

When items from secondary datasets are used, the qualification may also include the name of the link. When specifying the name of an explicit link, use two periods to separate the link name from the item name. For example, all of the following are valid:

```
L-NAME..INVOICE-DATE
L-NAME..INVOICE-HEADER.INVOICE-DATE
L-NAME..FINDB.INVOICE-HEADER.INVOICE-DATE
```

More information on links may be found in the *AskPlus Reference Manual* (<CHNAME_ASKPLUS_SELECTION>), *Visimage Reference Manual* (Chapter 4, Selection) and in the *Visimage Administrator Manual*.

Sub-items

A sub-item is a portion of an item. It is defined by an offset, a type and a length. A sub-item is usually used to extract a portion of a character field, but in certain special circumstances, it can also be used with other data types. A simple example of a sub-item may be used to extract the year and month portion of a date, provided the date is a character field. (This example assumes that the date is stored as CYMD.)

```
ORDER-DATE.(5:X4)
```

If the date is stored in a numeric field, it would be necessary to convert it to a character field and then use GETSTR instead of the sub-item syntax:

```
!GETSTR(!FORMAT("99999999",ORDER-DATE),"5,4")
```

The complete sub-item syntax is:

```
ITEM.(offset:type length)
```

Where:

Offset	The starting position measured in bytes (characters) of the sub-item being extracted, starting with position 1.
Type	The type of data that should be extracted in that position. The most common type is X, which indicates character data. The type can be omitted if it is the same as the type of the item. In other words if you are extracting character (X) data from a character (X) item, it is not required to specify the type.
Length	The length of the sub-item being extracted. The length is specified according to the type. In other words, for character (X) data, the length is measured in bytes (characters). For integer data it is measured in words and for packed data in nibbles (half a byte). See *Appendix B* for a complete list of types and lengths.

While the name sub-item indicates that this syntax is valid only for items, sub-items can be used with other operands also. Sub-items are valid for items, variables and system variables. Sub-items are NOT valid with constants, macros, special operators, or other expressions. The special operator !GETSTR (see below) can be used to accomplish the same effect as a character sub-item for operands that do not support the sub-item syntax.

Examples of Sub-items

The following are all valid sub-items:

```
DATE.(1:x4)
DATE.(1:4)
LAST-NAME.(1:x20)
PHONE.(4:Z7)
FILLER.(3:I2)
```

The last example shows an extraction of an integer field (I2) from an item called FILLER. In some databases, several different items are all concatenated together to form a single database item. If you know the offset, type and length of these fields you can extract them using sub-item syntax. It is best to define these fields in a dictionary (or macro), so that the complexity can be hidden from users. This same syntax can be used to extract fields from a file:

```
FILENAME.(11:P16)
```

The above syntax will extract a P16 field that starts in the eleventh byte of the record. This syntax is sometimes called CASTING as opposed to converting. Using this syntax, AskPlus interprets the data it finds in that location according to the rules for the specified type of the sub-item and not according to the type of the item. In other words, if FILLER is a character field and the sub-item specifies an integer type, AskPlus expects to find integer data not character data. Use the !CVT operator (see below) to convert data between different types.

If you are using a AskPlus dictionary and you use the same sub-item often, it is a good idea to ask the database administrator to create a new item in your dictionary.

Qualification of Items and Datasets

There may be cases when the name of an item, or even a dataset is not unique. This happens when the same item name is used in different datasets or the same dataset name is used in different databases. If you use a name that is not unique, it is best to fully qualify the name with the name of the dataset and/or the database.

If you do not fully qualify the name, AskPlus will prompt with a list of choices and ask you to select the one you want to use. To fully qualify an item name, use:

```
DATASET.ITEM   or
DATABASE.DATASET.ITEM
```

To fully qualify a dataset, use

```
DATABASE.DATASET
```

8.4 Constants, Wildcards, User Prompts & Variables

A constant is a number or character string whose value is always the same (constant). Constants are usually enclosed in “double quote” characters, but quotes are not always required. Numeric constants do not need quotes. Character constants require quotes if the value contains any lower case letters, blanks or special characters. Characters not enclosed in quotes will be automatically upshifted.

Examples of constants:

```
123
SMITH
“John Smith”
```

Wildcard (Smart) Constants

A *Wildcard* Constant is a special form of a character constant. Wildcards are most commonly used in the selection criteria, but may also be used in other logical conditions. Wildcards are only used in logical expressions involving **equals** or **not equals**. Wildcards allow a comparison to be based on a pattern of data rather than on an exact match. Wildcard constants can only be used to compare with **character** data. A wildcard value contains one (or more) of the special characters @,?,# or \. Each of these characters has a special meaning described below.

Wildcard characters:

Character	**Replaces**
@	Any character or number of characters.
?	A single alphanumeric character (letter or digit).
#	A single numeric character (digit).
\	Any single character.

Using Wildcards in AskPlus

Whenever a *Wildcard* character is used in AskPlus, a special syntax is required to tell AskPlus to treat the value as a wildcard value and not just a normal constant. This may be done in one of two ways:

- Using the SEARCH keyword in the FIND command. This keyword means that

all constants that use one of the wildcard characters in the current FIND command should be treated as a wildcard value.

- Add (S) or (SMART) immediately after the wildcard value. This syntax forces the current constant to be treated as a wildcard value.

Let's look at some examples:

Find all orders in 1994 that occurred on the thirteenth day of the month:

```
FIND SEARCH ORDER-DATE   =   "94##13"
```

Find all customers with the name JOHN anywhere in their first name:

```
FIND FIRST-NAME   =   "@JOHN@"(S)
```

Wildcard characters may also be used in a print condition:

```
D1, DESCRIPTION, TAB1, IF (STATUS   =   "A@"(S))
```

User Prompts

In many cases, you will need to create expressions where the value of one of the operands is specified by the user at run-time. This is most common in the selection criteria. To specify a user prompt, prefix the name of a prompt with a question mark character. When AskPlus executes the task, the user will be prompted to specify the value to be substituted in the expression. The name of the prompt (the text following the ? character) must follow the rules for variable names (see below). For example:

```
?PROMPT
?DATE-RANGE
?COLLATERAL_LIST
```

The selection chapter contains more examples of user prompts.

Variables

Variables are another type of operand. Variable names must follow the following rules:

- Must start with a letter (a-z, A-Z)
- Have a maximum length of 20 characters
- Cannot contain blanks or other special characters
- Can contain letters, numbers or one of the following:
  + - * / ? ! ‘ # % & @ $ _

Variables are discussed in more detail in the Report chapter.

8.5 Special Operators

Contrary to what their name implies, Special Operators are actually operands that can be used in an expression. They are called operators because they receive one or two operands as parameters and they perform an operation on the operands which returns a result. AskPlus contains a number of pre-defined special operators which are described in this section. It is possible for sites to create their own special operators. Contact your system administrator or technical support for more information. Depending on its purpose, a special operator can be used anywhere that you can use an expression.

Description of Special Operators

The syntax for using a special operator is:

```
!NAME:typelen(parm1, parm2)
```

The character ! is required and must precede the name. It signifies that what follows is the name of a special operator.

NAME

Designates the name of the special operator.

:typelen

Is the type and length of the value returned by the special operator. In most cases it is optional, however, it is a good practice to include it because it ensures that AskPlus knows what type of return value to expect. In the discussion of each special operator which follows, **typelen** will only be documented when it is required.

Parm1 and Parm2

Are parameters (operands) being passed to the special operator. A special operator can include one or two parameters. The parameter can be a simple operand or an expression. Below we will see examples where the parameter to one special operator is another special operator.

Working with Dates

The following special operators are used with dates. In most cases you will combine the operators !DATEXT and !DATINT to convert dates from one format to another. Expressions involving dates are great candidates to be saved as macros. Examples are shown after all of the date operators have been discussed.

!DATINT

Converts a **Date** stored in the specified **Format** into the AskPlus internal date format. The internal format is a UNIX standard, the number of seconds elapsed since January 1, 1970. Once dates are in internal format they can be reconverted to a variety of external formats using !DATEXT. It is also much easier to do date arithmetic with dates in internal format. The general syntax is as follows:

```
!DATINT(Date,”Format”)
```

Where:

Date	Specifies the date to convert to internal format. This date must be stored in the format specified by the Format parameter.
Format	Defines the format of the Date parameter, must be one of the following values:

Storage Format	**Description**
YMD, MDY, DMY	Date is stored using 6 digits without the century. Date Windowing (see below) is automatically supported. Dates stored using HP A0 date format (see MM3000 below) are also automatically supported. Date field may be numeric or character.
CYMD, MDCY, DMCY	Date is stored using 8 digits with the century included. Date field may be numeric or character.
CAL	Date is stored in HP3000 Calendar format. (Valid only in the range 1902 to 2027)
PHDATE	Powerhouse internal date format.
MM3000	A Y2K date format used by HP MM/3000. In this format, the date is stored in a 6 character field, but the century and the year are saved in the first two digits using letters for dates >= 2000. Also known as A0 dates.
Other	Most other date formats can be handled, by converting the date arithmetically to one of the other supported formats.

Examples:

```
!DATINT(ORD_DATE,”CAL”)
!DATINT(ENTRY-DATE,"CYMD")
```

!DATEXT

Converts a date in **internal** format back to an **external** format. The syntax is:

```
!DATEXT:typelen(Int_Date,”Format”)
```

Where:

`typelen`	Is the type and length of the result, not required.
Int_Date	Is a Date in internal format. Can be an expression involving !DATINT.
`Format`	Is one of the following output/display formats.

Display Format	**Description**
YMD, MDY, DMY	A 6 character date without the century.
CYMD, MDCY, DMCY	An 8 character date with the century.
CAL	HP Calendar format (valid from 1902 to 2027).
JUL	The Julian day within the year. January 1 is 1, December 31 is 365 (366 in a leap year).
WEEKNUM	Number of the week in the year (as described by ISO standard). Weeks are numbered from 1 to 52 or 53. Weeks begin on a Monday and the first week of the year is the week containing the first Thursday. When a year begins with a Friday, Saturday or Sunday, those days belong to the last week of the prior year.
WEEKDAY	The number of the day in the week (Sun=0, …, Sat=6)
DAYNAME	The name of the day.
MONTHNAME	The name of the month.
YEAR	The 4 digit year.
MONTH	The month number (1-12).

MONTHDAY	The day of the month (1-31).
MM3000	New Y2K format used by HP MM/3000. In this format, the date is stored in a 6 character field, but the century and the year are saved in the first two digits. Also known as A0 dates. (See below for more details.)
FDAYWEEK	The date of the first day (Sun) in the week. A 6 character date without the century.
LDAYWEEK	The date of the last day (Sat) in the week. A 6 character date without the century.
FDAYMONTH	The date of the first day in the month. A 6 character date without the century.
LDAYMONTH	The date of the last day in the month. A 6 character date without the century.
FDAYWEEK8	The date of the first day (Sun) in the week. An 8 character date with the century.
LDAYWEEK8	The date of the last day (Sat) in the week. An 8 character date with the century.
FDAYMONTH8	The date of the first day in the month. An 8 character date with the century.
LDAYMONTH8	The date of the last day in the month. An 8 character date with the century.

Examples:

```
!DATEXT:X6(!DATINT(ORD_DATE,”CAL”),”MDY”)
!DATEXT(!DATINT(ENTRY-DATE,"CYMD"), "FDAYMONTH8")
```

Date Windowing

Using *date windowing*, any two-digit year less than a certain threshold value is assumed to be in the 21 century while dates greater than or equal to the threshold are consider part of the 20 century. Thus if the threshold is 50, values from 00 to 49 will be treated as 2000 to 2049 and values of 50 to 99 will be treated as 1950 to 1999.

AskPlus will apply date windowing automatically whenever the system variable HPSPLITYEAR has been set. AskPlus uses HPSPLITYEAR as the threshold value and in the above example; HPSPLITYEAR would be equal to 50.

If you are using date windowing, be careful with birth dates which may suddenly appear to be in the future instead of the past. For example a birth date of April 5, 1927 which is stored as “270405”, will be interpreted as April 5, 2027 using date windowing.

MM3000 Format

In this format, dates are stored in 6 character fields, but year 2000 dates are stored using a letter code. Thus the years 2000 to 2009 are stored as A0 to A9 and the years 2010 to 2019 are stored as B0 to B9. This format supports dates up to 2250 to 2259 which are stored as Z0 to Z9. The advantage of this format is that dates will sort correctly without any conversion.

!SECOND

Converts an expression, which represents a number of minutes, hours, days or weeks, into seconds. This operator is used to add or subtract a number of units (usually days) from a date in internal format. The meaning of the value contained in **Expression** is determined by the value of **Unit**. The syntax is:

```
!SECOND(Expression,”Unit”)
```

Where:

Expression	Number of Units to convert to seconds.
Unit	Indicates the meaning of the Expression parameter. Must be one of the following characters:

Unit	**Meaning of Expression**
"M"	Minutes
"H"	Hours
"D"	Days
"W"	Weeks

Example:

```
!SECOND(NUM_DAYS,”D”)
```

The system variable $SECONDS is today’s date in internal format. It is very useful for date calculations involving today’s date.

!CVTSEC

The opposite of SECOND, this operator converts a number of seconds into minutes, hours, days or weeks. This operator is often used to determine the number of days between two dates. Once the dates have been subtracted, the result can be converted back into days or some other unit. The syntax is:

```
!CVTSEC:P16(Seconds,”Unit”)
```

Where:

P16	Is the return typelen and should be specified.
Seconds	Number of seconds to convert to the type specified by Unit.
Unit	Indicates the meaning of the result of using CVTSEC. Must be one of the following characters:

Unit	**Meaning of Expression**
"M"	Minutes
"H"	Hours
"D"	Days
"W"	Weeks

Examples:

The number of seconds in a day is 86400. Therefore, the following two expressions produce the same result.

```
!CVTSEC(expr,”D”)
expr / 86400
```

!WORKDAYS

Calculates the number of workdays (days not including Saturday and Sunday) between two dates. This operator does not take into consideration statutory holidays when calculating workdays. The syntax is:

```
!WORKDAYS(Date1, Date2)
```

Where:

Date1	The first date in internal format.
Date2	The second date in internal format.

Example:

```
!WORKDAYS($SECONDS,!DATINT(ORD-DATE,”CYMD”))
```

!DATEOR

Converts a date from the specified format into an Oracle formatted date. The syntax is:

```
!DATEOR(Date, "Format")
```

Where:

Date	The date in some external format.
Format	Defines the format of the Date parameter, must be one of the following values:

Storage Format	**Description**
YMD, MDY, DMY	Date is stored using 6 digits without the century. See !DATINT for more details.
CYMD, MDCY, DMCY	Date is stored using 8 digits with the century included. Date field may be numeric or character.

Example:

```
!DATEOR("20000101",”CYMD”) will return the value 01-
JAN-00
```

!DATEJOR

Converts a date from the specified format into an Oracle Julian date. The syntax is:

```
!DATEJOR(Date, "Format")
```

Where:

Date	The date in some external format.
Format	Defines the format of the Date parameter, must be one of the following values:

Storage Format	**Description**
YMD, MDY, DMY	Date is stored using 6 digits without the century. See !DATINT for more details.
CYMD, MDCY, DMCY	Date is stored using 8 digits with the century included. Date field may be numeric or character.

Example:

```
!DATEJOR("980603",”YMD”) will return the value 2450968
```

Examples of date conversions

The following examples show you how to use date conversion operators. We recommend that you create macros for your most common date conversions.

- Add 30 days to a date in CYMD format and display the result in MDY format:
  ```
  !DATEXT:X6(!DATINT(DATE,‘CYMD’) + !SECOND(30,”D”),
  ‘MDY’)
  ```
- Convert a date from MDY to YMD format:
  ```
  !DATEXT:X6(!DATINT(DATE,”MDY”),“YMD”)
  ```
- Subtraction of a number of days from a date and display it in CYMD format:
  ```
  !DATEXT:X6(!DATINT(DATE,“CYMD”) -
  !SECOND(DAYS,“D”),“CYMD”)
  ```

  DATE is an item formatted as CYMD and DAYS is a number of days to subtract from it. The result is a date formatted as CYMD.
- Calculate a person’s age in years using the item DOB (date of birth):
  ```
  ($SECONDS - !DATINT(DOB,“CYMD”)) / 86400 / 365.25
  ```

  This calculates the number of days between today and the person’s birthday and then divides by the number of days in a year. This gives an accurate age in most cases, but may not be exact if the current date is the same date as the person’s birthday. This next example, while more obscure, is more accurate. (This example presumes the birth date (DOB) is in “CMYD” format.)

  ```
  ($YEAR - DOB.(1:X4)) -
  (((!cvt:z2($MONTH) splice !cvt:z2($MONTHDAY)) <
     DOB.(5:X4)) / $TRUE)
  ```

  The first part of the expression subtracts the current year from the birth year. This is correct if the individual has had their birthday. If their birthday has not occurred, we must subtract one. This is accomplished by subtracting a relational expression divided by $TRUE. If the expression is true, we subtract one. If it is false, we subtract zero.

Working with Arrays

The following operators are used with arrays.

!SUM

Sums (totals) the specified array according to the rules designated by the second parameter. The syntax is:

```
!SUM(Array,”Start,End[,Increment]”)
```

Where:

Array	The name of the array item or variable to be summed.
Start	The first index (element) to be summed.
End	The last index to be summed.
Increment	The increment between Start and End. 1 means every element, 3 means every third and so on. The increment is optional. If no increment is specified, all values between start and end will be summed.

The last parameter must contain at least two values separated by a comma. It can be a constant or an expression involving SPLICE.

Examples:

Assume that `SALES` is an array item containing sales data for each of the 12 months of the year.

```
!SUM(SALES, "1,12,1")
```

Adds the values of SALES, from subscript 1 to 12, in increments of 1.

```
!SUM(SALES, “1,” SPLICE month)
```

Adds the values of SALES from subscript 1 to the value contained in the item or variable month in increments of 1.

See the discussion of Array Variables in the *Visimage Reference Manual* (Chapter 5, Creating Reports) for more examples.

!INDEX(@)

Is used with an array calculations involving the @ character. Its value is the current subscript of the array variable contained in the same calculation expression. The syntax is:

```
!INDEX(@)
```

Where:

@	This operator **always** takes the parameter @.

Examples:

```
ARRAY(@) = !INDEX(@)
```

This very simple expression will load one into the first element of array, two into the second and so on.

Finding Maximum and Minimum Values

A more complex example uses arrays to find the maximum and minimum values in an array. For this example, assume that an array SALES contains sales figures for each of the months of the year. Our goal is to find the maximum and minimum sales values.

To find the maximum value we use a temporary array called MAX which is initialized to the SALES array. We then walk through the array looking at each value and replacing it with the current maximum value. When we get to the end of the array, the last element will be the maximum value. For example:

```
#MAX(@) = #SALES(@)
#MAX(@) = #MAX(!INDEX(@)-1), IF
   (!INDEX(@) > 1 AND #MAX(!INDEX(@)-1) > #MAX(@))
```

Let’s look at this more closely. The first statement loads all SALES values into the MAX array. The second statement loads each element of the array, with the previous element of the array **only** if the previous element is larger than the current element. This means that the current element is always the maximum value and when we reach the end of the array MAX(12) is the maximum value. Notice that this statement is not executed for the first element of the array because there is no prior value.

The exact same approach will work to find the minimum value, just change the test from greater than to less than. However, finding the minimum is a little more difficult if you want to ignore zero values. To ignore zeroes use:

```
#MIN(@) = #SALES(@)
#MIN(@) = #MIN(!INDEX(@)-1), IF
   (!INDEX(@) > 1 AND (#MIN(@) = 0 OR
    (#MIN(!INDEX(@)-1) <> 0 AND #MIN(!INDEX(@)-1) <
#MIN(@)))
```

In this case we use the previous value if the current value is zero or the previous value is not zero and it is less than the current value. Only if one of these statements is true does the previous value qualify as the current minimum.

!GETINDEX

Searches the **Array** looking for the first occurrence of **Expression**. This operator returns the array index where the expression was found. Returns **zero** if the value is not found in any array element. The syntax is:

```
!GETINDEX:I1(Array,Expression)
```

Where:

:I1	You should always specify a return typelen of `I1` when using !GETINDEX.
Array	The array item or variable to search.
Expression	!GETINDEX will evaluate this expression and search for the result in the Array.

Example:

```
#IDX = !GETINDEX:I1(#ARRAY,"3")
```

If Array contains the values loading in the previous example, the value of IDX will also be 3.

Finding Maximum and Minimum Values

In the discussion of !INDEX(@) we showed how to find the maximum or minimum values of an array. These calculations located the correct values, but did not determine which element contained the maximum or minimum value. This can be determined using !GETINDEX.

```
#MAX-IDX = !GETINDEX:I1(#SALES,MAX(12))
#MIN-IDX = !GETINDEX:I1(#SALES,MIN(12))
```

!GETMAX

Returns the index of the part of the array specified that contains the maximum value. Does not return the maximum value itself, but rather the first index of the element containing the maximum value.

Using this operator is much easier than the examples shown above.

The syntax is:

```
!GETMAX:I1(Array,"Start,End,Increment")
```

Where:

Array	The name of the array item or variable to be checked.

Start	The first index (element) of the array to check.
End	The last index (element) of the array to check.
Increment	The increment between Start and End. 1 means every element, 3 means every third and so on.

This last parameter must contain all three values separated by commas without any embedded blanks.

!GETMIN

Same as GETMAX, but returns the index of the minimum value of the array values specified.

The syntax is:

```
!GETMIN:I1(Array,”Start,End,Increment”)
```

!EXTRACT

Returns the element of the array specified by the idx_expr.

The syntax is:

```
!EXTRACT:typelen(array, idx_expr)
```

This operator is obsolete since an expression can be used directly as the index to an array.

String Operators

Two of these operators are very useful when working with character fields. The first, !STRPOS, returns the **starting position** of a character or string in a string. The second, !GETSTR, will extract a sub-string from within a string. !GETSTR is similar to using a sub-item with two important advantages. It can be used with an expression, and the offset and length can be variables.

!STRPOS:I1

This special operator returns the starting position of a find string within a specified source string. If the find string is not contained in the source string, !STRPOS returns the value zero. For example, !STRPOS:I1(“ABCD”,”B”) will return the value 2 which is the position of “B” in “ABCD”. The general syntax is as follows:

```
!STRPOS:I1(source_str, search_str)
```

Where:

:I1	Is the type and length (Integer 1) of the result returned by !STRPOS. You must always specify :I1 when using this special operator.
source_str	Is the source string that you want to search through looking for the first occurrence of find_str. Source_str can be any character field or expression that returns a character result.
search_str	Is the value of the string you are searching for. Search_str can be a single character, a character field or any expression that returns a character result. !STRPOS will return the position of the **first** character of the **first** occurrence of search_str.

Examples:

```
!STRPOS:I1(MEMBER-NAME, “/”)
!STRPOS:I1(BILL-ADDRESS-3, STATE.(1:x2))
!STRPOS:I1(LAST-NAME, “JR.”)
```

More examples will be given following the discussion of GETSTR.

!GETSTR:Xn

This special operator extracts a sub-string from a source string. The sub-string is extracted based on the position and length. For example, `!GETSTR:X2(“ABCD”,”3,2”)` will return the value “CD”. This special operator is more powerful than normal sub-item syntax because it allows all of the parameters to be expressions.

Use !GETSTR when the source string is an expression, or when the position or lengths are variables. The syntax is as follows:

```
!GETSTR:Xn(source_str, “pos, len”)
```

Where:

:Xn	Is the type and length returned by !GETSTR. You must always specify a type of X. The length can be any integer between 1 and 256.
source_str	Is the source string from which you want to extract the sub-string.
pos	Is the starting position of the sub-string, where the first character is position 1. The pos and len parameters must be enclosed in quotes and separated by a comma.

len	Is the length of the sub-string to be extracted. The pos and len parameters must be enclosed in quotes and separated by a comma.

Even though the syntax requires that the values for the pos and len parameters must be enclosed in quotes and separated by a comma, it is still possible to calculate a variable starting position and length. For example, if you have two variables called pos and len, you can specify `#pos splice "," splice #len` *as the second parameter to !GETSTR.*

Examples:

```
!GETSTR:X20(MEMBER-NAME, “1,” splice #len)
```

```
!GETSTR:X4(!CVT:z8(ORD-DATE),”1,4”)
```

The first example extracts the first **len** characters of member name.

The second example converts an integer date stored in CYMD format into a zoned 8 and then extracts the first four characters (Century and year). See below for more examples.

Examples of !STRPOS and !GETSTR Working Together

Imagine that you wanted to find all records that contained “JR.” in the last name and then print the portion of the last name that occurs before “JR.”. We will up-shift the name before looking so that we catch all spellings of JR. Assume that LAST-NAME is an X30 field in the set CUSTOMERS:

```
Selection Criteria:
!STRPOS:I1(!SHIFT:X30(LAST-NAME, “U”), “JR.”) <> 0
```

```
#LEN = !STRPOS:I1(!SHIFT:X30(LAST-NAME, “U”),”JR.”) -
1
#SHORT-NAME = !GETSTR:X30(LAST-NAME, “1,” splice #LEN)
```

The above example starts by selecting all records where JR. is found somewhere within the last name. If STRPOS returns a non zero value, then the find string was found somewhere. Then in the report, the variable LEN is loaded with the starting position of JR. minus one. This is the length of the name up to, but not including, JR. Finally, the portion of the name from 1 to LEN is extracted.

Let’s consider another example where the item NAME contains the member’s last name followed by the first name. The first and last names are separated by a slash character. We want to print the first name followed by the last name with a single space in between. If no slash is found, the name will not be changed.

```
Variables:
   POS:I1
   NEW-NAME:X30

Calculations:

   #POS = !STRPOS:I1(NAME, ”/”)
   #NEW-NAME = !GETSTR:X30(NAME,
         (#pos + 1) splice “,” splice (30 - #pos))
join
         !GETSTR:X30(NAME, “1,” splice (#pos - 1)), IF
#POS > 0
   #NEW-NAME = NAME, IF #POS = 0
```

The above example looks for a slash in the item name and returns its position in the variable pos. Then a new name is calculated based on the position of the slash. This calculation is only done if POS is greater than zero (i.e. the slash was found). The calculation joins together two sub-strings extracted using GETSTR. The first one returns the portion of the name that follows the slash (from POS+1 to the end). The second one returns the first part of the name (from 1 to POS-1). Finally the new name is loaded with the original name if a slash was not found (POS = 0).

Summit customers should use the macro %MEM-FULL-NAME or %CVT-FULL-NAME to perform this calculation automatically.

!SHIFT

Converts all the characters of the **String** parameter to upper or lower case. The direction (upper or lower) of the shift is determined by the second parameter. The syntax is:

```
!SHIFT(String,”Mode”)
```

Where:

String	A character expression to be converted to upper or lower case
Mode	The case to be shifted to:

Mode	**Action**
“U”	Shift to UPPER case.
"D"	Down - Shift to lower case.
"I"	InitCap - First letter of each word is upper case.

Example:

```
!SHIFT(FIRST-NAME, "U") = "JOHN"
```

In this example, the item FIRST-NAME will be up shifted **before** comparing it to the value JOHN.

!FORMAT

This special operator allows you to apply a format (edit mask) to an item or expression and then use the result in another expression. Traditionally, formats could only be applied when printing an item or expression to your printer or screen. The special operator !FORMAT allows the formatted value to be downloaded to PC applications and to be used in other expressions. The syntax is as follows:

```
!FORMAT("format", expression)
```

Where:

format	Is the format (edit mask) to be applied to the expression. The format must be enclosed in quotes.
expression	Is the item or expression to be formatted.

Examples:

```
!FORMAT("$$$,$$9.99-", AMOUNT / 100)
!FORMAT("xx/xx", DATE.(1:x4))
```

For example, the result of a call to !FORMAT might be used with !GETSTR to retrieve a specific text value ($1,000). When downloading to a file, !FORMAT can be used with output to a host or local file to download a formatted number as a character string rather than as a numeric value. This is useful if you want to do a mail merge and you want the formatted amount to appear in a letter.

Miscellaneous

The remaining operators fall into no special category.

!COMPRESS

Removes all spaces from the **String** parameter. This operator takes only one parameter. The syntax is:

```
!COMPRESS(String)
```

Where:

String	Any character expression.

The result of using !COMPRESS is to remove all banks and compress the remaining characters together.

!CVT

Converts its parameter (expression) to the type and length defined by typelen. The syntax is:

```
!CVT:typelen(Expression)
```

Where:

Typelen	Specifies the type and length of the result, the value of the parameter will be converted to that type and length. Any of the types and lengths found in Appendix B may be used.
Expression	Any VISIMAGE expression. The operator !CVT takes only one parameter.

Example:

If the item CODE contains the value 3456, then

```
!CVT:Z8("12" CAT CODE)
```

Gives the result 00123456 in zoned format with leading zeros.

!EVEN

Returns either -1 or 1 depending on whether or not the value of **Expression** is even (1) or odd (-1). The syntax is:

```
!EVEN(Expression)
```

Where:

Expression	Is any numeric expression. If the value of the expression is even, the result is 1. If it is odd, the result is -1.

!RANGE ((Cond), “Parm”)

This operator is used **only** in the **Selection Criteria** window. It is used to select records from a primary file as a function of the number of times that a condition is matched in a secondary file. The syntax is:

```
!RANGE((Condition),”Parm”)
```

Where:

Condition	Is a logical condition involving a secondary topic. The condition

	must be enclosed in parentheses.
Parm	Is the range parameter.

See the Selection chapter for an example using Range.

!SIGNED

Forces the sign of a Packed (P) or Zoned (Z) **Expression**. Packed and Zoned data can have three different signs, negative, positive and unsigned. This operator only affects unsigned values by changing them to positive. The syntax is:

```
!SIGNED(Expression)
```

Where:

Expression	Is any expression returning packed or zoned data.

!TITLE

Returns the default title associated with an **item**. This operator is used internally by VISIMAGE to print default column headings in list. It is rarely used directly by a user. The syntax is:

```
!TITLE(Item)
```

Where:

Item	Is the item name which may be fully qualified.

8.6 System Variables

Contrary to what their name implies, system variables usually have a constant value. They are called variables, because their value can vary between successive runs of a task. However, during the execution of a task, the value of most system variables is constant. One notable exception is $PAGENO whose value changes on every page. AskPlus includes a number of predefined system variables that are described below. Their types and lengths vary as a function of the information they contain. Depending on its purpose, a system variable can be used anywhere that you can use an expression.

Description of System Variables

The syntax for using a system variable is:

```
$NAME
```

The character `$` is required and must precede the name. It signifies that what follows is the name of a system variable.

NAME

Designates the name of the system variable.

The following system variables are available in AskPlus:

Name	Typelen	Description
$ACCOUNT	X8	The name of the logon account (HP e3000).
$DATE	X8 or X10	Today's date formatted with a two-digit year. Normally same as $DATE6 unless the system variable DATE8 equals 1. In this case $DATE will be the same as $DATE8. $DATE and $DATE6 will display Y2K years using HP's MM3000 format unless the system variable ASK_OLDDATE has been set to 1. The order of YY/MM/DD depends on the setting of DATEFORMAT.
$DATE6	X8	Today's date in 6 digits (two character year). Default format is MM/DD/YY.
$DATE8	X10	Today's date in 8 digits (four character year). Default format is MM/DD/YYYY

Name	Typelen	Description
$DAYNAME	X10	The name of today’s day.
$ENTRIES	I2	The number of entries selected in the most recent selection.
$FALSE	K1	The constant FALSE.
$HOMEGROUP	X8	The name of the logon group (HP e3000).
$INFO	X132	The INFO string passed to the ASKPLUS program. (HP e3000)
$JULIANDAY	I1	Today expressed as a Julian day.
$MISSING	K1	Special value for data selection
$MONTH	I1	The current month expressed as a number (1-12).
$MONTHDAY	I1	The current day of the month expressed as a number (1-31).
$MONTHNAME	X10	The name of the current month.
$PAGENO	I1	The current page number of the list or report.
$PARM	I2	The PARM value passed to the ASKPLUS program (HP e3000).
$SECONDS	P16	Today’s date and time in **internal** format.
$SESSION	X8	The name of the current session (HP e3000).
$SYSDATE	I1	The current date in HP (calendar) format.
$SYSTIME	I2	The current time in HP (clock) format.
$TIME	X8	The current time in the form HH:MM AM/PM.
$TRUE	K1	The constant TRUE.
$USER	X8	The current logon user name (HP e3000).
$WEEKDAY	I1	The day of the week corresponding to today expressed as a number (1-7, where Sun = 1).
$YEAR	I1	The current year in the form CCYY.

Logon Information

The variables $USER, $ACCOUNT, $HOMEGROUP and $SESSION are all character fields that contain the current logon information. Only $USER is valid on UNIX systems, the rest are valid only on the HP 3000.

Current Date and Time

Many of the system variables provide information about the current date and time in a variety of formats. For Wednesday July 4, 2001, 10:15 AM the system variables will have the following values:

$DATE	01/07/04 (International date format)
$DATE6	07/04/01 (US date format)
$DATE8	07/04/2001 (US date format)
$DAYNAME	WEDNESDAY
$JULIANDAY	185
$MONTH	7
$MONTHDAY	4
$MONTHNAME	JULY
$WEEKDAY	5
$YEAR	2001
$TIME	10:15 AM

The value of $DATE depends on the setting of the DATEFORMAT command. See the *Visimage Administrator Manual.*

Other variables represent the date and time in internal formats that do not make sense when printed.

$SYSDATE	This is the date in HP calendar format. See the *HP Intrinsics Manual*.
$SYSTIME	This is the time in HP clock format. See the *HP Intrinsics Manual*.
$SECONDS	The current date and time in internal format. The number of seconds since January 1, 1970.

The variable $SECONDS is very useful for performing date calculations.

For example yesterday's date in CYMD format is:

```
   !DATEXT:X8($SECONDS - 86400,”CYMD”)
```

Where 86400 is the number of seconds in a day.

Miscellaneous Variables

The remainder of the variables do not fit into any specific category.

- The variables $TRUE and $FALSE are constants. See Visimage Reference Manual (Chapter 7, Examples) for some interesting examples that use $TRUE.
- The variable $ENTRIES is the number of entries that qualified in the most recent selection. This variable is sometimes used to print a message when no entries have been qualified. The message is printed conditionally if ($ENTRIES = 0)
- Use $MISSING as a special constant in the selection criteria to determine if there are any matching records in a secondary set. See the *Selection Chapter* for more information.
- Use $PAGENO to print the current page number on every page.
- The variables $INFO and $PARM contain the INFO and PARM values passed to AskPlus.

8.7 Macros

A macro is a name given to a complex expression. It is a shortcut for calculations that are repeated frequently. The AskPlus Administrator creates and maintains macros and makes them available.

See the *Visimage Administrator Manual* for more information on creating macros. The return types and lengths vary as a function of the information they contain. Depending on its purpose, a macro can be used anywhere that you can use an expression.

Description of Macros

The syntax for using a macro is:

```
%MACRO(parm1, parm2, ...)
```

The character `%` is required and must precede the name. It signifies that what follows is the name of a macro.

NAME

Designates the name of the macro.

Parm1, Parm2 …

The macro parameters. Macros are not required to have parameters. However, if the macro definition contains parameters, the user must provide the correct number.

9 Running & Configuring AskPlus

- Running AskPlus
- Batch Mode
- Configuring AskPlus

9.1 Running AskPlus

The commands to start AskPlus depend on the host system. The basic syntax is described below, followed by optional run time parameters that may be included as part of the run command.

MPE (HP 3000) Run Command

On MPE, AskPlus is normally installed in the ASKPLUS account. The most recent version of AskPlus will be in the PUBTCP group. Older versions of AskPlus may be found in the PUBNAT group, however, these versions do not contain all of the latest features. Refer to the *Visimage & AskPlus Installation & Configuration Guide* for more information on installing AskPlus. The basic run command is:

```
RUN ASKPLUS.PUBTCP.ASKPLUS;XL="XL.PUBTCP.ASKPLUS"
```

For convenience, "LIB=G" may be used instead of the "XL=" parameter. The following POSIX syntax is also valid:

```
RUN /ASKPLUS/PUBTCP/ASKPLUS;XL="/ASKPLUS/PUBTCP/XL"
```

AskPlus should always be started from a colon prompt and not from within the POSIX shell.

UNIX Run Command

On UNIX, the AskPlus software may be installed in any location chosen by the system administrator. During the installation, a symbolic link called ASKPLUS is created that refers to the AskPlus installation directory. Refer to the *Visimage & AskPlus Installation & Configuration Guide* for more information on installing AskPlus. The following command will start AskPlus, regardless of where it was installed, because the command uses the symbolic link. Remember that case is important on UNIX systems.

```
/ASKPLUS/askplus
```

Specifying a Visimage Profile Name (-a)

When AskPlus is started it is possible to specify a default environment that should be used. The environment may be specified using a Visimage Host Profile name. Information on creating Visimage profiles is located in the *Visimage & AskPlus Installation & Configuration Guide*. The profile name is specified using the "-a" option. AskPlus will inherit the exact same default environment as Visimage, including the working directory, except for the login parameter. The examples below show AskPlus being started with a profile name called "demo".

MPE (HP 3000) Example:

```
RUN ASKPLUS.PUBTCP.ASKPLUS;INFO="-ademo";LIB=G
```

MPE run line parameters are specified using the INFO string. For simplicity, this example uses "LIB=G" instead of the "XL=" parameter.

UNIX Example:

```
/ASKPLUS/askplus -ademo
```

There must be no spaces between the option "a" and the profile name "demo".

Note: The vsrvtcp listener process must be running for AskPlus to process the "-a" option.

The default port (service) used by vsrvtcp may be specified using the "-n" option.

Specifying a Manager File (-u)

Another way to specify a default environment is to specify the name of a manager file. In the following example, the manager file is called vmgrdemo.

MPE (HP 3000) Example:

```
RUN ASKPLUS.PUBTCP.ASKPLUS;INFO="-uvmgrdemo";LIB=G
```

UNIX Example:

```
/ASKPLUS/askplus -uvmgrdemo
```

If both "-a" and "-u" are specified, the "-a" option will take priority and the "-u" option will be ignored. If the vsrvtcp listener is not running, then the "-u" parameter will be used.

Specifying a Command (XEQ) File (-xeq:)

It is possible to specify the name of an command file that will be executed immediately when AskPlus is started. Unless the command file contains an EXIT command, AskPlus will remain running after the command file has been executed. On MPE systems, the filename must be entered in file.group.acct syntax (case is not important). On UNIX systems, the file name is case sensitive. For example:

MPE (HP 3000) Example:

```
RUN ASKPLUS.PUBTCP.ASKPLUS;INFO="-xeq:cmdfile.xeq";LIB=G
```

UNIX Example:

```
/ASKPLUS/askplus -xeq:../xeq/cmdfile
```

9.2 Running AskPlus in Batch (Script)

When AskPlus is executing in batch (or in a UNIX script) it cannot prompt the user for additional information. In batch, AskPlus will use the following defaults instead of prompting the user:

- When qualifying ambiguous items or sets, AskPlus will use the first available choice in the list. We strongly recommend that you fully qualify all ambiguous item and set names to avoid having AskPlus select the wrong item (or set) by default.
- When creating files (and neither DELETE or APPEND was specified), AskPlus will use DELETE as the default. Any existing file of the same name will be deleted.
- When resolving user prompts (?PROMPT), AskPlus will assume that the next line in the script contains the value and will read the value from the script. If the prompt is inside an XEQ file, AskPlus will read the next line of the script (not the next line of the XEQ file) to get the value.

Use Visimage (File > Execute Offline > Create a New Job) to create a sample job. The sample will show you the basic syntax for the host system that you are currently using.

By default, AskPlus will abort if an error occurs while executing in batch.

MPE (HP 3000) Jobs

On MPE, the first character of any line containing an AskPlus command must not be an ! (exclamation). If, for example, your selection criteria begins with !RANGE, insert a blank character before the exclamation. This will prevent the MPE operating system from interpreting the ! as a command prefix and changing it to a colon.

On MPE, AskPlus will automatically know that it is running in a job.

UNIX Scripts

On UNIX, the "-b" parameter must be specified to "tell" AskPlus that it is running in batch. If this parameter is omitted, AskPlus will attempt to prompt the user for input (for example when qualifying an ambiguous item) and will expect the next line of the script to contain a valid response. Sample run command:

```
/ASKPLUS/askplus -b << 'EOD'
various AskPlus commands
EXIT
```

```
EOD
```

Other options, such as -aprofile, may also be specified prior to the "<<".

The syntax `<< 'EOD'` is used to "tell" UNIX that the commands which follow (in the script) should be passed as input to the AskPlus program. All commands, up to the specified label (in this case EOD) will be passed to the program. If single quotes are not used with the label, UNIX will perform $variable substitution before passing the input lines to AskPlus. If single quotes are used (as in the example), UNIX will not modify the lines in any way. We recommend using single quotes as shown, unless you need the $variable substitution. In this case, any AskPlus statement that uses a $ ($DATE or "$$$,$$9.99-") must be escaped with a '\' (back slash). For example \$DATE or "\$\$\$,\$\$9.99-".

Working with UC4

When using UC4, the first (non blank) character of any line in the job or script must not be an ! (exclamation). If, for example, your selection criteria begins with !RANGE, you must insert ":DATA " (and a blank character) before the exclamation. This will force UC4 to process the statement as valid code and not treat it as a comment.

Summary of Run Command Options

The following table summarizes all of the options discussed above.

Option	Meaning
-aprofilename	Specify the name of a Visimage host profile to use when starting AskPlus.
-b	"Tells" AskPlus that it is running in batch. UNIX only.
-nport	Specifies the vsrvtcp port name/number.
-ufilename	Specifies the name of a manager file to establish the default environment. This parm is ignored if "-a" is also used.
-xeq:filename	Specified the name of an XEQ file to execute immediately after startup.

9.3 Configuring AskPlus

Certain AskPlus functions can be modified by setting appropriate system variables. The following table lists the variables that can be set and their meaning. On MPE these variables can be set in the CMDIN file. On UNIX, these variables can be set in the /ASKPLUS/askplus script. Variables can also be set in the vsrvtcp INI file. Variables set in the INI file will be used for all Visimage connections and also for AskPlus when started with the "-aprofile" parameter.

VAR Name	Value	Has the following meaning for ASKPLUS and VISIMAGE
ABORT_BATCH_ ON_ERROR	Y	Force AskPlus to abort in batch if an error occurs. UNIX only. (MPE always aborts.) See also Assign Batch.
ALLOWMULTISE TDELETE	Y	Allow records to be deleted from more than one set (in the select file.) (Default - multi-set deletes are not allowed.)
ASK_OLDDATE	1	Display today's date ($DATE, $DATE6) using "00", "01" instead of "A0", "A1" for Y2K dates. (Default – use "A0", "A1")
ASKC	1	Accept all ASKC syntax. (Default – some ASKC syntax is supported.)
ASKCCTL	1	On UNIX, force CCTL (form feeds) to be generated in all report output - both file and printer. Overrides the OUT= option.
ASKERR	1	Limit the same error message to print a maximum of 50 times. (Default – no limit.)
ASKJOINS	1	For relational databases, use joins stored in the dictionary.
ASKODX	1	Links will use an IMAGE key in priority over an IMSAM key. For performance, this should always be activated. (Default – Use the IMSAM key.)
ASKORADATE	1	Check with ARES.
ASKPATH0	Path	Specifies the full path to locate the ASKPLUS program files. MPE only. (Default: "PUBTCP.ASKPLUS")
ASKSEL	1	Create/use a select file compatible with ASK Version C. (Default – ASKC select files are not recognized.)
DBCLOSEMODE2	1	Close all datasets between commands. This should always be activated. (Default – datasets are not closed.)

VAR Name	**Value**	**Has the following meaning for ASKPLUS and VISIMAGE**
EQ_DBSERVER	ipaddr: port	Specifies the location of the Eloquence database server.
FORCEDIE	1	Always terminate, never suspend. MPE only. (Default – ASKPLUS will suspend if possible.)
GAUGE	1	Activate ASKPLUS percentage messages during serial reads. (Default – no percent messages.)
HPSPLITYEAR	nn	This value determines how to treat YMD dates. Two digit dates less than nn are treated as 2000 dates. Dates >= nn are handled as 1900. (Default is 50.)
IEEE	1	Assume that all R2 and R4 fields in ALL databases contain HP format real numbers. (Default - IEEE format is assumed.)
IEEEbase	1	Assume that all R2 and R4 fields in the specified base contain HP format real numbers. (Default - IEEE format is assumed.)
KSAMNAT	1	Create compatibility mode KSAM files. MPE only. (Default – Native mode KSAM files are created.)
LOGONFILESON LY	1	Restricts the OPEN command to open files and databases that are located in the users logon ACCOUNT (directory).
MAX_IMAGE_IT EMS	1024	The maximum number of Image/Eloquence items in the database. You must increase this value if your db has more items.
OMNIDEX	1	Ignore any Omnidex or Suprdex keys that are found. (Default – Omnidex and Suprdex keys are used.)
ORACLE_HOME	path	The path to Oracle's installation directory.
ORACLE_SID	"A"	The Oracle SID.
ORACLE_VERSI ON	nn	The Oracle version number.
PACKED	nn	The default size to create PACKED variables. Range is 20 – 28.
PH60	1	Assume that Quiz sub-files are in version 6.0 or higher. MPE only. (Default – Quiz sub-files are in version 5.0)
PROTECT	1	Prohibit users from accessing commands that modify the database. (Default – These commands are available in ASKPLUS, but not in VISIMAGE.)

VAR Name	**Value**	**Has the following meaning for ASKPLUS and VISIMAGE**
REPBLANK	1	ASKPLUS will fill the end of a report with blank lines when the report is created with OUT=filename. This guarantees that the file is an exact number of pages long.
SECURITY-MSG	File	Specifies the name of a security message file that will be included at the front of all output from the REPORT command.
SH_LIBPATH	path	The path for the shared library files. UNIX only.
SHOW_CONNEC T_INFO	1	Show additional connection info when AskPlus is started with the "-aprofile" option.
SHOWERR	1	Display more extensive file system errors. (Default – standard messages are displayed.)
SRVTRACE	1	Create the SRVTRACE tracing file.
STARLIST	1	Display error messages from reports created using STARJET.
SUPR_QUEUE	CS,DS ,ES	The queue to use when starting SUPRTOOL as a son process. Default value is DS.
SUPRPROG	prog	The name of the SUPRTOOL program (MPE) to start. Default value is SUPRTOOL.PUB.ROBELLE.
TRACE_SQL	1,2	1 – Display the SELECT statement to STDLIST 2 – Write the SELECT statement to a file. (Default, SELECT statement is not displayed or written to a file.)
TZ	value	Sets the current time zone. See HP documentation. For example, use "PST8PDT" for Pacific Time.
VMGRLIST	1	Echo the Manager File in VISIMAGE tracing window. MPE only. (Default – no echo.)

A APPENDIX A

A.1 Writing a user exit procedure

A *user exit* is a procedure written, compiled and prepared by the user, that is completely integrated into AskPlus at execution time. User exits are designed to provide user specific functionality not included in AskPlus. A programming error in such a procedure can make AskPlus abort. Thorough testing is required before putting it into operation.

On MPE, your procedures must be stored in the log-on group SL, in the log-on account PUB group SL, or in the system SL. On UNIX they must be included in a shared library.

This information assumes that you are familiar with the steps required to write, compile and load a user written procedure.

Description

A *user exit* procedure may be written in any language able to generate an SL or shared library procedure.

These procedures are called *user exit* because they receive data, they modify it, and they return the new data. What they return is decided by the programmer.

A *user exit* works with a maximum of 3 operands: 2 input and one output result. Each operand defines 3 parameters in the calling sequence (operand, type, length). So, a *user exit* procedure has 9 parameters, all of them referenced by address. The operand is a byte array, the type is one byte and the length is an integer (See *typelen* in Appendix B).

Calling sequence

The calling sequence in SPL is defined as follows:

```
procedure proc ( op1, type1, len1,     << 1st  operand
>>
```

```
      op2, type2, len2, << 2nd  operand >>
      op3, type3, len3 );     << result >>
      byte array op1, op2, op3;
      byte type1, type2, type3;
      integer len1, len2, len3;
```

All parameters, except 'op3' are input parameters. Normally, only 'op3' is supposed to return something to AskPlus. The result of the operation performed by the procedure is to be returned in 'op3', in the type/length specified by 'type3' and 'len3'.

However, if you need to change the type/length of the returned value (type3 and len3), you can do it, providing that the new length in bytes does not exceed the previous one. Beware : Since there are different ways to receive parameters in a procedure (depending on the language), it is necessary to specify it if it is not SPL. The following syntax must be used :

```
   !PROCNAME[:typelen][ :lang ] (parm1, parm2)
```

where *lang* can take the following values : SPL, FTN (Fortran), COB (Cobol), PAS (Pascal). Default is SPL.

If all *user exits* of a *report* are written in the same language, you may set a JCW :

```
   :SETJCW USEREXIT,0         << SPL >>
   :SETJCW USEREXIT,1         << COB >>
   :SETJCW USEREXIT,2         << PAS >>
   :SETJCW USEREXIT,3         << FTN >>
```

If the first parameter value of a *user exit* is in 'X10', the second one in 'I2' and if the returned value is in 'P12', the Cobol calling sequence must be defined as follows :

```
LINKAGE SECTION.
01 arg1 pic x(10).
01 type1      pic xx.
01 len1 pic s9(4) comp.

01 arg2 pic s9(9) comp.
01 type2      pic xx.
01 len2 pic s9(4) comp.

01 arg3 pic s9(11) comp-3.
01 type3      pic xx.
01 len3 pic s9(4) comp.

procedure division using
  arg1, type1, len1,
  arg2, type2, len2,
  arg3, type3, len3.
```

B APPENDIX B

B.1 AskPlus Data Types & Lengths

The data types supported and used by AskPlus are the same as those used by the TurboImage and Eloquence databases.

Numeric Types:

There are many different numeric data types, but they can be divided into two different groups: integer and floating point. Integer data types hold whole numbers with no decimal portion. Numbers stored in floating point data types include the decimal portion.

Signed Integer (I & J)

The integer type is the most common data type for storing integer numbers. Integer values may be represented by the letter I or J. The storage format is the same. The I type is most commonly used and J is not generally used. The number is stored as a signed (2's compliment) value. The length represents one 16 bit (2 byte) word. The valid lengths are I1 (J1), I2 (J2) and I4 (J4). Avoid using I1 as a counter since the maximum value is only 32,767. Both I2 and I4 are capable of storing large numbers.

TypeLen	**# of Bytes**	**Approximate Value Range**
`I1 (J1)`	`2`	`+/- 32,767`
`I2 (J2)`	`4`	`+/- 2 billion`
`I4 (J4)`	`8`	`+/- 8 billion billion`

Unsigned Integer (K)

Unsigned integers (positive whole numbers) are represented by the letter K. The most common use of this type in AskPlus is to represent a conditional or logical type. Unsigned integers are stored using 2's compliment with no sign bit. The length represents one 16 bit word. The valid lengths are K1, K2 and K4.

TypeLen	**# of Bytes**	**Approximate Value Range**
`K1`	`2`	`0 to 64,000`

K2	4	0 to 4 billion
K4	8	0 to 16 billion billion

Packed Integer (P)

The packed type supports a larger range of integer values than is provided by I4. Packed values are represented by the letter P. Each digit in a packed number is stored using a nibble (4 bits) and the sign character also occupies one nibble - a 16 bit word will hold a 3 digit number. The length represents one nibble (1/2 byte). The length must be an even number from 2 through 28.

TypeLen	**# of Bytes**	**Value Range**
P2	1	+/- 9
P4	2	+/- 999
...	...	...
P28	14	+/- a 27 digit number

Floating Point (R and E)

The floating point types store numbers using scientific notation which includes the sign a mantissa (decimal portion) and an exponent. Floating point numbers can be extremely large (or small), however they are not guaranteed to be 100% accurate. (The storage format does not support all possible numbers.) The HP Real type is represented by the letter R. The IEEE type is represented by the letter E. These two types use a different method to store the mantissa and the exponent. The length represents a 16 bit word. Valid lengths are R2, E2 and R4, E4. Since the larger types provide more accuracy, we recommend creating floating point types using R4 or E4.

TypeLen	**# of Bytes**	**Accuracy (*)**	**Approximate Value Range**
R2	4	6.9	+/- 8.6E-78 to 1.27E77
R4	8	16.5	+/- 8.6E-78 to 1.27E77
E2	4	7.2	+/- 1.4E-45 to 3.4E38
E4	8	17.9	+/- 4.9E-324 to 1.8E308

(*) The number of decimal digits of accuracy.

Character (String) Types:

All of the character types allocate a single ASCII character to each byte. The length of all of these types represents the number of bytes used to store the value:

Character Data (X)

The most common character type is represented by the letter X. Each character contains a single ASCII character. The maximum length of a character string is 1023 characters.

TypeLen	**# of Bytes**	**Description**
Xn	n	a string of up to n characters

Character Data (U)

This data type is intended to store UPPER case alphabetic characters only. In actual fact, a U type string may contain any ASCII character.

TypeLen	**# of Bytes**	**Description**
Un	n	a string of up to n characters

Zoned (Numeric) Data (Z)

The zoned type (represented by the letter Z) is a character string that contains only digits. Each byte contains a single digit and the sign character is over-punched in the last digit. Leading zeroes are always stored as zero (not blank). Zoned data may be used with string concatenation operators (like a string) and with arithmetic operators like a number. The maximum length of a zoned field is 255 bytes.

TypeLen	**# of Bytes**	**Description**
Zn	n	a number of up to n digits

Logical Types:

No specific logical type is provided with the TurboImage/Eloquence databases. With AskPlus, use the type K1 for creating logical (True/False) variables.

TypeLen	**# of Bytes**	**Value**
K1	2	True (1 - non zero), False (0)

Date Types:

No specific date type is provided with the TurboImage/Eloquence databases. The most common method for storing dates is to use a character field (X8) or an integer field (I2).

TypeLen	**# of Bytes**	**Value**
X8	8	CYMD using ASCII digits
I2	4	CYMD as a numeric value

Index