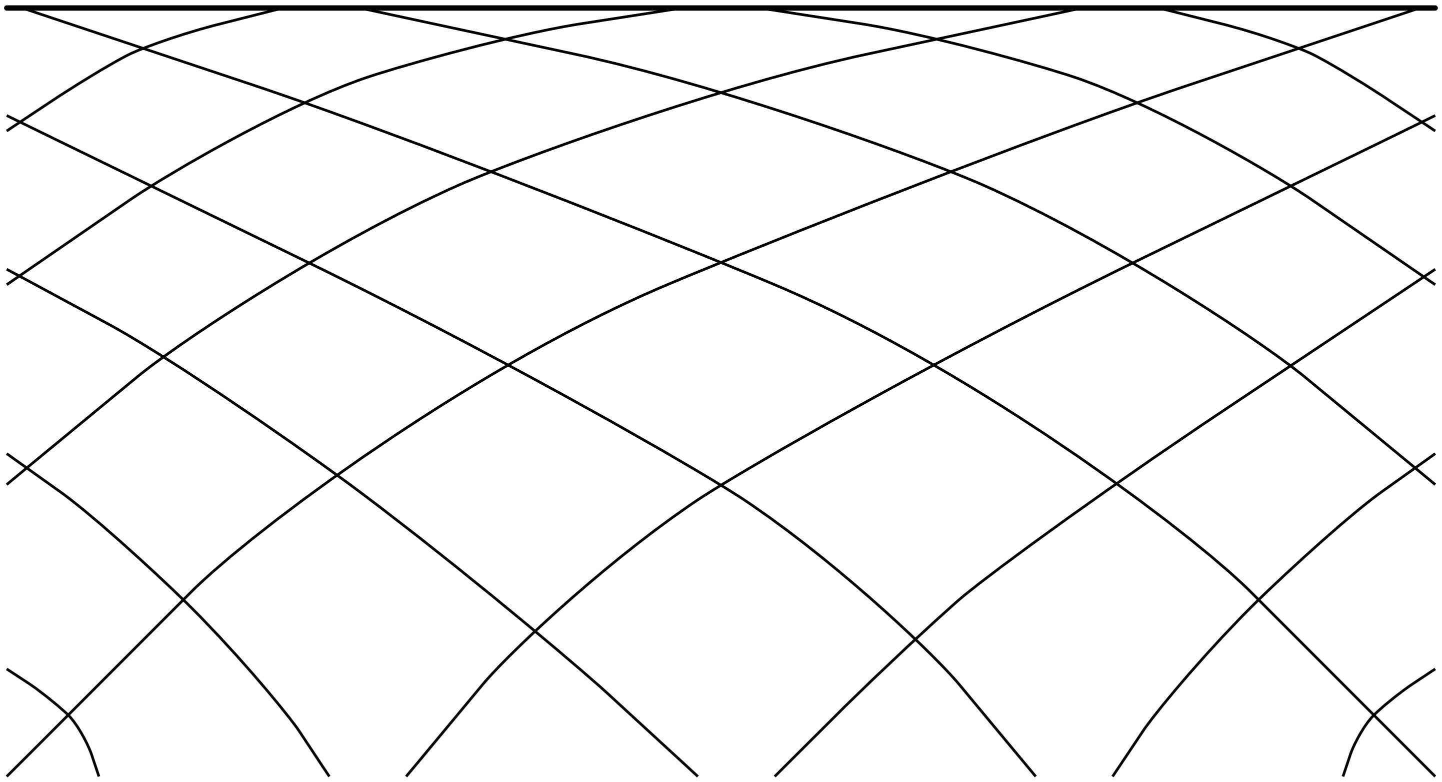
585-223-703
Issue 1
March 1990

DEFINITY®Manager IV
Query and Report Languages



TO ORDER COPIES OF THIS DOCUMENT

CALL:	AT&T Customer Information Center (800) 432-6600 In Canada: (800) 255-1242
WRITE:	AT&T Customer Information Center 2855 North Franklin Road P.O. Box 19901 Indianapolis, Indiana 46219

For more information about AT&T documents, see
Business Communications Systems
Publications Catalog (555-000-010)

Prepared by
AT&T Technical Publications Department
Lincroft, NJ

CONTENTS

ABOUT THIS GUIDE

The DEFINITY® Manager IV Query Language is used to gather information to supplement the standard reports found in the Manager IV applications Terminal Change Management (TCM) and Facilities Management (FM). The Report Language allows you to format the results of these queries in an easy-to-use and flexible language. The Manager IV Query and Report Language Manual describes how to gather and format information from the Manager IV database and display and print it in an attractive format. This manual also includes queries and reports drawn from the Manager IV applications.

The first four sections of this manual form a tutorial for using the Query and Report Languages, and the last two sections serve as the Command Directory. If possible, try to read the first section in order. This tutorial should provide the information necessary to begin the Manager IV Query and Report Languages.

About This Guide	This preface describes the audience and conventions of this manual.
Section 1 The Query and Report Languages	This section describes the relationship between the Query and Report Languages and Manager IV.
Section 2 The Manager IV Database	This section describes the database structure and provides the information about the Manager IV files and fields that will enable you to use the query language most effectively.
Section 3 The Core Terminal Monitor	This section describes the commands of the Core Terminal Monitor (CTM), the environment in which queries are created and executed. This section also describes how to use help to get lists of the files and fields that are found in the Manager IV database.
Section 4 Developing Queries	This section explains the commands of the Query Language itself and includes examples of queries drawn from TCM and FM situations.
Section 5 Generating Reports	This section describes the process involved in generating reports and the commands and syntax of the Report Language and includes several sample reports drawn from the Manager IV applications TCM and FM.
Section 6 CTM and Query Language Commands	This section contains the command directory for the Core Terminal Monitor (CTM) and Query Language and describes each command, its syntax and an example of its usage. This section includes commands from the CTM as well as the query commands because they are used together.

Section 7 Report Language Commands	This section contains the command directory for the Report Language and describes each command, its syntax and an example of its usage. It includes report language commands, text formatting commands, as well as the commands needed to compile and print reports.

Audience

This manual is intended for the System Administrator who will be the primary user of Query and Report Language. The System Administrator should have some knowledge of the UNIX® operating system and a screen editor such as **vi** or **emacs**.

Conventions Used In This Manual

In this manual, the following conventions are used:

- **Bold** type is used to represent commands.
- *Italic* type is used to represent user-determined input, such as a filename.
- Messages and system prompts are shown between quotation marks.
- This manual uses examples extensively. Examples are usually preceded by a heading labeled "Example."
- **RETURN** is used to represent the RETURN key on your keyboard.

1. THE QUERY AND REPORT LANGUAGES

The Manager IV Query Language enables you to gather information about the records that are stored in the Manager IV database. You can gather this information by using Query Language commands to develop, test and execute queries that you have developed. The Query Language is used to search the Manager IV database. While the Manager IV applications have many standard reports that meet the needs of most situations, the Query Language allows you to customize your request, and get the results quickly without having to run a scheduled report. The query text can also be integrated into a report and the format can be selected using the Manager IV Report Language.

Queries are usually developed and written by the System Administrator, who has database responsibilities, and who should have some knowledge of the UNIX operating system. Requests for queries and reports will come, most likely, from the TCM or FM application administrators, or their management. The actual queries will be developed and tested by the System Administrator, and then, this same query text can be used to generate a report using the commands of the Manager IV Report Language.

Queries are developed using the Core Terminal Monitor (CTM), a program used to create, test and execute queries. Queries can also be developed in the UNIX file and then run under CTM if the user prefers. The resulting output from the queries may not be nicely formatted, but the query text can be integrated into a formatted report using the Manager IV Report Language. The source files for reports are prepared using a UNIX screen editor such as **vi** or **emacs**. These source files are then compiled and run in the UNIX shell to create reports. Queries and reports can be saved as UNIX files and can be edited, printed and rerun as needed.

The Manager IV Query and Report Languages enables you to produce customized reports whenever you want or need them. If one of Manager IV's standard reports does not meet your exact needs you can develop the query and format the results to produce the report that you need.

You can easily prepare templates for creating reports in the style that you want by developing sample report code for the format that you want, and leaving the section where the query belongs blank. You can simply insert the query text into the template to run the report. You do not have to prepare new report source code each time you want to run the report. Of course, you can create as many of these templates as you want.

The first four sections of this manual form a tutorial for using the Query and Report Languages, and the last two sections serve as the Command Directory. If possible, try to read the first four sections in the order in which they appear in the manual. This tutorial should provide the information necessary to begin using the Manager IV Query and Report Languages.

2. THE MANAGER IV DATABASE

The Manager IV database should be considered a relational database. In the Manager IV database, information is stored in files in the form of records.

The Manager IV database is composed of one or more subfiles, called targets, which contain information about a particular product or corporation. Although all Manager IV files are stored in one database, the database files are logically divided by subfile. You must specify a target (subfile) when you use the query language (just as you would using one of the Manager IV applications), but you can access more than one target at a time. If you are querying a Number Portability file, you must also specify the Number Portability ID.

The Manager IV Query Language provides statements and commands that enable you to gather information from the Manager IV database quickly and efficiently. You define specific queries to get just the information that you want. The Manager IV Report Language provides statements and commands that enable you to report the information you have gathered clearly and understandably.

With Manager IV, you also have access to detailed on-line database information about the file and fields of the Manager IV database through the Query Language **help** command. This command provides specific information about the files and the fields found in your Manager IV site that you need to construct queries.

Database Files

The Manager IV database files are listed alphabetically and are described in the table that follows. Depending on your site configuration, you may have some of the files and not have some others. This is a generalized list.

This table also includes information that will be necessary in constructing the query. Most of the files listed in the table require a standard target, usually a PBX identifier (the name given to the product), but some files require another kind of target. For example, Number Portability files require that the Number Portability identifier be used as their target. One of the fields in the file (Target) lists the type of target required to query that particular file. The choices are the following:

- Prod (the product name assigned at initialization)
- Corp ID (assigned at initialization)
- NP ID (assigned at initialization)

Table 2-1. Manager IV Files

File Name	Controlling Application	Target	File Description
aar	FM	prod	Automatic Alternate Routing Patterns. Each record specifies one preference for an Automatic Alternate Routing (AAR) pattern.
aca	FM	prod	Automatic Circuit Assurance (ACA). Contains ACA referral information polled from the prod's ACA audit trail. [System 85/DEFINITY Generic 2 only]
adgrp	TCM	prod	Abbreviated Dial Groups. Each record specifies one abbreviated dial group and the characteristics of that group. [System 85/DEFINITY Generic 2 only]
alias	ALL	fixed	Contains additional information for products defined in the cust file.
ap	TCM	prod	Application Processor. Each record specifies one AP and the services available on that AP.
ars	FM	prod	Automatic Route Selection Patterns. Each record specifies one preference for an Automatic Route Selection (ARS) pattern and plan.
arscc	FM	prod	Automatic Route Selection (ARS) Call Categories. [System 85/ DEFINITY Generic 2 only]
arsclk	FM	prod	ARS 7-Day Clock. Each record specifies one ARS plan number and the prod time to that ARS plan.
auth	FM	prod	Authorization Codes. Each record specifies one authorization code and the characteristics of that code.
autovon	FM	prod	AUTOVON NNXD Routing Patterns. Each record specifies one AUTOVON NNXD routing pattern.
casrlt	FM, TCM	prod	Centralized Attendant Service - Release Link Trunks. Each record specifies one CAS branch, the branch type, and the release link trunk group associated with that CAS branch.
ccg	TCM	prod	Call Coverage Groups. Each record specifies the characteristics of one call coverage group.
ccgp	TCM	prod	Call Coverage Group-Points. Each record specifies the characteristics of one call coverage group point.

Table 2-1. Continued

File Name	Controlling Application	Target	Description
cdr	FM	Sys 85 DEFINITY G2	Call Detail Record format specifications. Each record contains the number of words and placement of information on the CDR for each
conti	FM	prod	Contact Interface. Each record specifies one contact interface circuit and the characteristics of that circuit.
cos	TCM	prod	Line Class Of Service. Each record specifies the characteristics of one line class of service.
crac	FM	prod	Code Restriction - Allowed Calls. Each record specifies one allowed numbering plan area (NPA) and/or one office code associated with one code restriction trunk group.
csmic	TCM	DIM FP8	Custom Intercom.
cust	ALL	fixed	Customer information. This file contains one record for each PBX, adjunct, NP Network and Corporation in the network.
dciupt	FM	prod	DCIU Port. Identifies the application associated with each DCIU port on a prod.
dciurt	FM	prod	DCIU Alternate Routing. Each record specifies the alternate routes associated with one destination map.
dcsnet	TCM, FM	corp	DCS Network. Each record specifies information associated with one DCS network.
desig	FM	prod	NPA/NXX Designator. Each record lists the designator assigned to each thousands digit/NPA-NXX combination.
dsconn	FM, TCM	prod	Dedicated Switch Connection. Each record contains information for one pair of dedicated prod connections.
ects	TCM	DIM	Each record contains the set attributes for one Electronic Custom Telephone System (ECTS) terminal.
ectsc	TCM	DIM FP8	ECTS Controller. Each record contains information about one ECTS controller.
equip	TCM, FM	prod	Equipment. Each record specifies one equipment location and the characteristics of that location.
etnnet	TCM, FM	corp	ETN Network. Each record specifies information associated with one ETN network.
extn	TCM	PBX	Extensions. Each record specifies the characteristics of one extension. Includes all extensions defined in the dial plan.
feadac	TCM, FM	PBX	Feature Dial Access Code.
fmdata	FM	prod	Facilities Management Data. Each record specifies information associated with one Manager IV database partition’s FM transactions.
fivdgt	FM, TCM	PBX	Route calls to local/remote PBX(s)

Table 2-1. Continued

File Name	Controlling Application	Target	File Description
fnpa	FM	prod	Foreign NPA (ARS). Each record specifies one 3-digit translation pattern number for one numbering plan area.
frl	FM	prod	Facility Restriction Level. Each record specifies one facility restriction level and its alternate facility restriction level.
fstdgt	TCM, FM	PBX	First Digit. Information associated with the First Digit Table for each PBX in the network. Also used to determine the call source.
hnpa	FM	prod	Home NPA (ARS). Each record specifies one 3-digit translation pattern number for one numbering plan area.
intercept	FM	prod	Each record specifies the intercept treatment for one intercept type.
links	FM	prod	DCIU Administration - Link Characteristics. Each record specifies the characteristics of one DCIU link.
mrgcos	TCM	prod	Miscellaneous Trunk Restriction Groups - Class Of Service. Each record specifies one class of service and the miscellaneous trunk restriction groups associated with that class of service.
mrgdac	FM	prod	Miscellaneous Trunk Restriction Groups - Dial Access Codes. Each record specifies one dial access code and the miscellaneous trunk restriction groups associated with that dial access code.
name	TCM	PBX	Names - User Data. Each record specifies information associated with one Manager IV user.
npextn	TCM	NP Net.	Number Portability Extensions. Each record specifies the characteristics of one number portability extension.
npnet	TCM, FM	corp	Number Portability Network. Each record specifies information associated with one number portability network.
npnode	FM	prod	Node and Call Category --> AAR Pattern. Each record specifies the AAR pattern number associated with one node number-call category pair.
nprnx	FM	PBX	RNX --> Node Or First Digit. Each record specifies the node number or first digit associated with one RNX.
nprout	FM	prod	First and Thousandth Digit --> RNX.
nrcall	FM	prod	Non-Restricted Calls. Each record specifies one office or area code that can be accessed by a toll-restricted, code-restricted, or ARS terminal.
pcc	FM	prod	Processor Communication Circuit.

Table 2-1. Continued

File Name	Controlling Application	Target	File Description
pq	TCM, FM Maint.	prod, NP Net.	Pending Queue. Each record contains information to be downloaded to the prod from Manager IV.
rdce	FM	prod	Restricted Dial Code Entries. Each record specifies one restricted dial access code.
rnx	FM	prod	RNX --> AAR Pattern Assignment. Each record specifies the AAR pattern number that is associated with one RNX.
rnxcc	FM	PBX	RNX And Call Category --> AAR Pattern. Each record specifies the AAR pattern number that is associated with one RNX-call category pair.
set	TCM	PBX	Set Attributes. Each record contains set attributes for one terminal.
sixnnx	FM	prod	6-Digit Translation (ARS). Each record specifies the ARS pattern number associated with one area code-office code pair.
smb	TCM	prod	Buttons. Each record specifies information associated with one button.
spdcl	TCM	DIM	Speed Calling. Each record specifies one Speed Calling group and its characteristics.
spq	TCM, FM, Maint.	prod, NP Net.	Shadow Pending Queue For each transaction record in the Pending Queue, the Shadow Pending Queue stores equivalent information that reflects the state of the product or Manager IV database prior to the update.
srfile	TCM, FM Maint.	corp	Service Requests Each record contains status information about pending service requests.
syscos	TCM, FM	PBX	System Class Of Service. Each record specifies the characteristics of one system class of service.
targrp	SysAdmin	corp	Each record contains information associated with one target-group.
tenant	TCM	Sys 85 DEFINITY G2	Each record contains information related to one extension partition used in Tenant Services.
tendgt	FM	prod	10-Digit And Unauthorized Call Control. Each record specifies one 10-digit conversion or one unauthorized call.
tollc	FM	prod	Toll Table - Direct Trunk Access.
tollt	FM	prod	Toll Table (ARS).
trkextn	FM	Sys 85 DEFINITY G2	Each record contains information related to one trunk group/extension partition used in Tenant Services.
trkgp	FM	PBX	Trunk Group. Each record specifies the characteristics of one trunk group.
trunks	FM	prod	Trunks. Each record specifies the characteristics of one

			trunk circuit.

Table 2-1. Continued

File Name	**Controlling Application**	**Target**	**File Description**
ucd	TCM, FM	prod	Uniform Call Distribution Groups. Each record specifies the characteristics of one UCD, ACD, or enhanced UCD group.
unterm	TCM	prod	Unassigned Terminals. Each record specifies the location and characteristics of one unassigned terminal.
vector	FM	prod	Each record contains information for one step of a vector used with the Call Vectoring feature.
wiring	TCM	prod	Wiring Assignments. Each record specifies information associated with one cable wiring assignment.
wcallcat	FM	prod	World Class Routing (WCR) Call Category. Each record specifies the characteristics associated with a particular Call Category.
wdialplan	FM	prod	WCR Dial Plan. Each record specifies the dialplan information associated with a particular Network.
wdigitmod	FM	prod	WCR Digit Modification. Each record specifies the digit modification parameters for a particular Digit Modification Index.
wdigitsend	FM	prod	WCR Digit Sending. Each record specifies the digit sending parameters for a particular Digit Sending Index.
wisdn	FM	prod	WCR ISDN. Each record contains the ISDN specific parameters for a particular ISDN Sending Index.
wnetroute	FM	prod	WCR Network Route. Each record defines mapping of Virtual Nodepoint Identifiers to WCR Routing Patterns for one Network.
wnetwork	FM	prod	WCR Network. Each record defines the "global" parameters associated with a particular Network.
wpattern	FM	prod	WCR Pattern. Each record defines the parameters associated with all preferences within a network routing pattern for a particular Network.
wtollfree	FM	prod	WCR Toll Free. Each record specifies the toll access codes associated with a particular Network.

3. THE CORE TERMINAL MONITOR (CTM)

The Core Terminal Monitor (CTM) is a program that is used to create, test and execute queries. It is your primary interface to the Manager IV Query Language. This section describes the most commonly used commands of the CTM and includes a sample session. It also describes how to use the CTM **help** command to list the files and fields of the Manager IV database.

ACCESSING CTM

Not all Manager IV users can access the Query Language, but the System Administrator can. If you have access to the login privileges of the System Administrator, you can access CTM easily from the UNIX shell by following this procedure.

Accessing CTM from the UNIX Shell

1. At the $ prompt, enter **ctm** and press **RETURN**.
   - The system will respond with the message "go" followed by an asterisk (*).
   - The asterisk is the prompt that you will receive while you are in CTM. You can begin entering queries now.
   - If you do *not* receive that message followed by an asterisk, you may see the following message: "sh:ctm not found." This indicates that CTM is not yet in your path.

Accessing CTM from the Manager IV Applications

You can also access CTM directly from any of the Manager IV applications by following this procedure.

1. From the Manager IV path line, type **shell create** and press **RETURN**.
   - You will see the message "data submitted-please wait" followed by the prompt "env, shell, or list >"
2. Enter **s** for shell and press **RETURN**.
   - The UNIX shell prompt $ will be displayed.
3. Type **ctm** and press **RETURN**.
   - You will see the word "go", followed by an asterisk, which is the CTM prompt. This indicates that the CTM query buffer is empty and ready to receive input.
   - If you do not receive the message "go", followed by an asterisk..... repeat the above steps.

Query Buffer

Within CTM, there is a **query buffer** that holds the current contents of a query. When CTM is ready to accept input, the message "go" is printed. This indicates that the query buffer is currently empty. If the message "continue" is displayed, this means that there is something currently in the query buffer.

CTM COMMANDS

There are several commands available in CTM. You can access the list of commands by using the command **\help**.

Generating a list of CTM commands

1. At the CTM "*" prompt, enter **\help** and press **RETURN**.

   - You will see the following list of all the available CTM commands:

```
append (a)
chdir <dir>
date (time)
echo ()
edit (e) <filename>
go (g)
help (h)
include (i) <filename>
print (p)
reset (r)
shell (sh) <command>
quit (q)
write (w)
```

The CTM commands are entered after the * prompt and preceded by a backslash (\). You need to enter only the abbreviation of the command, which is found in the parentheses, after the backslash. Some commands such as **edit** and **include** can be used with a filename.

We will discuss the most commonly used commands and give some examples of their usage.

Commonly Used Commands

go (g)

The **go** command executes the current contents of the query buffer. Enter **\g** after a query and press **RETURN** to execute the query.

help (h)

The **help** command is used to display the commands used in CTM. To get more specific information about a particular command use **\help** followed by the name of the command, for example, **\help help** would display information about the help command itself.

write (w)

The **write** command is used to save the current contents of the query buffer. Follow the command with the filename you wish to use. When you are saving a query, follow the filename with a **.q** to distinguish it from other files. This will become particularly important later when you include the query files into report code. If you have edited an existing file, you must also remember to end the editing session with the write command or the changes will not be saved.

The format of the write command is **\w** *filename***.q**. Choose whatever filename you wish to identify the query.

print (p)

The **print** command is used to display the current contents of the query buffer to the screen. It cannot be used to generate printouts. However, once a query file has been saved, it is a UNIX file and you can use the standard UNIX commands (such as **cat**) to print it. The format of this command is **print**.

You can use this command to check the query you have written. The filename is assigned in the preceding **write** command.

edit (e)

The **edit** command is used to change a query that has already been saved to a file. A UNIX editor such as **vi** or **emacs** is automatically invoked when the \ command is given, so knowledge of vi is required. If you are editing a file that is currently in the query buffer, there is no need to specify a filename. If you wish to edit a file that is not in the query buffer, follow the edit command with the name of the file you wish to edit. You may save the changes to this file, but this will not change the contents of the query buffer.

Use this command when you want to correct an error or change an existing query. Choose whatever filename you wish and remember to append the **.q** to it.

include (i)

The **include** command is used to append the contents of a file onto the query buffer. This is particularly useful if the current CTM process is interrupted for some reason. The syntax of the include command is **\i** *filename***.q** (where *filename* is the name of a stored file). After you "include" a file into the query buffer, you should use the **\print** or **\edit** commands to display the file.

chdir (cd)

the **chdir** command is used to change from your current directory to the directory specified in the command. CTM uses the current directory to store its temporary files, and if the current directory is full, or cannot be written to, you may wish to change to another directory without exiting CTM.

quit (q)

The **quit** command is used to terminate the CTM session and get back to the UNIX shell prompt. If you have not saved the contents of the query buffer, then you will lose what you have written. Remember to use the **write** command to save files and queries.

reset (r)

The **reset** command erases the contents of the query buffer. The former contents of the query buffer are lost and cannot be retrieved. This should be used if many mistakes have been made writing a query and you want to clear out the buffer and begin again.

Here are some other CTM commands that are less frequently used but may be helpful to you.

append (a)

The **append** command is used to add lines to the query buffer. Typing **\append** after executing a query allows you to add lines to the end of the query without entering edit mode (by default, the query buffer is reset after a query is executed).

date

The **date** command prints out the current date. This is particularly useful when you are writing reports because this command will automatically record the date that you are executing the query. The format of this command is **\date** .

time

The **time** command prints out the current time. This is particularly useful when you are writing reports because this command will automatically record the time that you are executing the query. The format of this command is **time** .

shell (sh) *UNIX command*

The **shell** command allows you to temporarily escape to the UNIX shell. This allows you to run one or more UNIX shell commands without having to exit from CTM. The contents of the query buffer are not affected if you exit to the UNIX shell this way.

The format of this command is **\sh** *UNIX command*. For example, from CTM, you could enter the command **sh ls** and the files in your directory will be listed. However, when the files have been displayed , you will be automatically back in CTM.

4. DEVELOPING QUERIES

This section describes how to develop queries in CTM and describes the syntax and commands of the Query language. This section also includes several examples of queries likely to be used for the Manager IV applications TCM and FM.

CREATING QUERIES: A CHECKLIST

There are several steps involved in creating and executing queries:

1. Receive a request for information, probably from the Facilities Administrator or the Terminal Change Administrator.
2. Enter the CTM environment by typing **ctm** at the UNIX shell prompt.
3. Use the Query **help** command to determine which files and fields need to be searched, or you can also use the command **ftpr** to print out a copy of your entire database schema with all the files and their fields identified.
4. Write the query in CTM.
5. Test the query using the CTM command **\go**.
6. Save the final query text using the CTM \write command. Append the suffix **.q** to the filename you choose for the query.
7. Leave CTM and get back in the UNIX environment using the CTM command **\quit**.

QUERY ENVIRONMENT

Queries are written, tested and executed in the CTM environment. CTM commands were described in Section 3. Once you enter CTM, and receive the " go" prompt you are ready to begin entering queries. The go prompt indicates that the query buffer is empty and ready to receive query text. When you save a query you have written with the CTM command **write**, that query will be saved in your current UNIX directory.

Help

You can find out what Manager IV files are resident on your system and also find out what fields can be found in each of those files by using help. This help is available through CTM. To find out what files are in the database, enter **help** and press **RETURN**. When you have located the file that you want, you can check to see what fields are available in this file. To do that, enter **help** *filename \g* where filename is the name of the file you want field information on. To display the fields of the extension (extn) file, you would enter **help extn \g** and all of the fields would be displayed.

This information must be known to write a query. Refer to Section 2 of this manual for the table of the files in the Manager IV database.

Printing the Database Schema

You can also use the command **ftpr** to print a copy of your database schema. Your schema is dependent on the options you have selected at your site, so the **ftpr** command gives you exactly what you have.

When you enter the command, you have the option of saving the output of the **ftpr** command to a UNIX file, which you can save or print. If you save the file, you can use the standard UNIX search commands (forward slash "/", followed by the string you are searching for, to search forward in the file, and "?", followed by the string you are searching for, to search backwards) to find the information you want quickly. You will probably want a copy of the schema (also called field table) to keep in a binder while you are working.

Sample Session

To get a copy of your database schema, follow these steps.

1. From the UNIX shell prompt, enter the command **ftpr**. Since you will most likely want to save the file, save the output of the file to a standard UNIX file so that you can search on line as well as retain a file to print at any time.
2. To save the file, enter the command **ftpr** followed by the name of the file you wish to save it as. For example, if you enter **ftpr > db.file** , this command will save the schema to a file called db.file. Now you can use the UNIX editor to quickly search the file for the particular files and fields that you need.

File Management for Queries

To keep track of queries you have written, you may want to keep all of your queries in the same directory. Use the UNIX command **mkdir** to create a directory for queries. Once you begin writing queries and integrating them into reports, you may find that your directory is crowded with too many similarly named files and you may not recognize the file type from its name. Therefore, we recommend that you append the suffix **.q** to queries, and the suffix **.rpt** to report source files. The name for compiled reports will be taken automatically by the system from the report file.

Sample Query Session

This brief example demonstrates how a query is developed in CTM.

```
$ ctm
*
go
* target is "9992233"
* range of e is extn
* retrieve (e.extn, e.setid, e.state)
* where e.state = "w"
* \g

* * executing ........
```

The Query Language Statements used to construct the above queries and all others are described in detail in the next section.

QUERY LANGUAGE STATEMENT TYPES

There are five statements that can be used to construct a query; three are required and two are optional.

- target
- range
- retrieve
- where (optional)
- sort (optional)

Each one will be described in detail below.

Target

The **target** statement is required. The target identifies the particular partition of the Manager IV database that you want to access. You can identify the target by either its LDN or a PBX identifier, which is an alphanumeric string such as "blue".

There must be at least one target statement; although, there can be more than one. Multi-target statements are particularly useful for querying Distributed Communications Systems (DCS) or Number Portability networks.

Example (One target)

```
* target is "9998833"     (LDN as target)
or
* target is "pegasus"     (PBXID as target)
```

Note that the LDN or PBXID is enclosed in quotation marks.

Example (Two targets)

```
*target t is "att"
*target p is (201)3334456"
```

As you can see from the example of two targets, the target needs a one-letter identifier to distinguish one target from the other. In the example above, the identifiers used in the target statements are **t** and **p**. The importance of this will become obvious when we look at another statement in the query language, **retrieve**.

Range

The **range** statement identifies the particular Manager IV database *file* or *files* that you want to access. This is a required statement and there may be more than one range specified per query. You can use **help** in CTM to check the names of your Manager IV files. Each file that is to be searched must be specified in this statement.

Example

```
* target is "9992233"
* range of e is extn
* range of eq is equip
```

You should precede the name of each file with an identifier. It can be a single or double letter identifier. It is usually the first letter of the file name, but if there are two file names being searched that begin with the same letter (extn, equip), you could use the first two letters of one of the files and the first letter of the other so that the identifiers can be easily distinguished.

In the example, there are two files that begin with the same letter (extn, equip). The identifier **e** is used with the extn file, and the identifier **eq** is used for the equip file. Or, you could use two letters for each: **ex** for extension, and **eq** for equipment.

The previous example contains an example with one target; now, we will show how the **range** statement differs in a query with two targets.

Example

```
* target t is "blue"
* target p is "9992233"
* range of a is acct in t
* range of e is extn in p
```

The only difference in the **range** statement with two targets is that a target identifier must be added to the **range** statement, in this case, t for the first target, and p for the second target. Otherwise, the range statement is the same as in a query with one target.

Retrieve

The **retrieve** statement identifies *fields* of the files that you have selected in the **range** statement you wish to view. This statement is required and you can have only one per query. However, you may specify several fields in this statement. The fields specified must be contained in the files you have named in the range statement. Again, you can use **help** in CTM to check this. To retrieve all fields in a file, the keyword *all* is used in place of the field name.

The single letter (or double letter) identifier in the range statement is used here to distinguish between fields that may be shared by more than one file. For example, the field **state** is found in both the extension (extn) and the equipment (equip) files and the identifier tells the system which file to search for that field.

The **retrieve** statement is not affected whether the query has one or more target statements because the **retrieve** statement specifies which fields in a particular file to search, and these files are linked to the particular Manager IV target in the **range** statement.

Example

```
* target is "9992233"
* range of e is extn
* range of eq is equip
* retrieve (e.extn, e.setid, e.state, eq.car)
```

In this example, information would be displayed from the four fields contained in the parentheses: extension (extn), set identification number (setid), state (state), and carrier (car). You can also see the importance of the identifier preceding the field name here because three of the fields are from the extension file and the fourth is from the equipment file. Because the identifier **e** has been selected for the extension (extn) file, you can see that the state field being queried is from the extension (extn) file.

Where

The **where** statement qualifies the query. This is an optional statement. There can be only one **where** statement in a query but it can be quite complex.

The **where** statement narrows the query by making the conditions for which you are searching quite specific.

Example

```
* target is "blue"
* range of e is extn
* range of eq is equip
* retrieve (e.extn, e.setid, e.state, eq.car)
* where e.state ="w"
```

State is a commonly found field in several Manager IV files and three states are recognized in Manager IV:

- working (w)
- available (a)
- reserved (r)

Working indicates that the equipment is in operation, *available* means it can still be assigned and *reserved* means that it is not available for administration.

The **where** statement is not limited to the state field. Logical operators can also be used in **where** statements.

Logical Operators

- and
- or
- not
- greater than >
- less than <
- equals =
- greater than or equal to >=
- less than or equal to <=
- not equal to !=
- outer join =^

Parentheses can also be placed around statements to control the order of precedence.

Example

```
* target is "9992233"
* range of e is equip
* range of s is setid
* retrieve (s.setid, e.equip, s.equip, s.mtype)
* where (e.equip = s.equip)
```

The above statement means that we want to display records where the values of the equipment fields from the equipment file (e.equip) match the equipment fields from the setid file (s.equip).

Arithmetic Operators

You can also use the following arithmetic operators to write queries.

- + to add
- - to subtract
- * to multiply
- / to divide

You can also use parentheses to control the order of precedence with the arithmetic operators.

Sort

The **sort** statement is used to sort the fields in the results of the query. You do not have to wait until you write a report to do this. However, sorting a large amount of data may take a long time. If you are working with a large file, you may want to run the query, and then from the UNIX shell, use the UNIX sort command on the query.

The default used with the sort command is ascending. This will alphabetize strings and put numbers into ascending order.

Example

```
* target is "9992233"
* range of e is extn
* retrieve (e.extn, e.setid, e.state)
* sort by setid
```

You do not need an identifier with the **sort** command, just specify the field name. To sort by descending order, you only need to change the **sort** order by entering **setid:d**.

Understanding the individual statements is not so difficult; neither is putting them together into a query. However, formulating queries from requests for information or reports may be a bit more difficult.

QUERY FUNCTIONS

There are several functions that can be used with the Query Language. They include the following:

- **Any**--to check to see if any record meets the qualification
- **Atof**--to convert strings to floating-point numbers
- **Atol**--to convert strings to integers
- **Avg**--to compute the average of values
- **Clip**--to remove trailing blanks from a field; therefore, changing the length of the field
- **Concat**--to join two fields together
- **Count**--to count the occurrences of a specified value
- **Left**--to access a left substring of a field
- **Right**--to access a right substring of a field

- **Max**--to compute the maximum value of a field
- **Min**--to compute the minimum value of a field
- **Sum**--to add values

We have chosen a few of the query functions to detail for you:

Any

Any is a function that returns a value of 1 if any record satisfies the qualifications of the statement or 0 if no record meets the criterion.

Atol/Atof

Most of the field values in the Manager IV database are stored as strings. However, you may want to query to find if there are working extensions greater than 2000 or within some range that you specify. To do this successfully, you should first convert the extension that is stored as a string to an integer using the function **atol,** or when a floating point format is needed, use the function **atof**. These functions should be used in conjunction with some of the other Query Language functions such as **count**, **avg** and **sum**.

Count

The function **count** can also be used in developing a query. This statement gives you the number of occurrences of a particular event. To use this function though, you must set a variable to represent the number of occurrences sought in the query.

In the following querynumber would give you the number of extensions in the working state. **Example**

Example

```
*target is "chicago"
* range of e is extn
* retrieve number = count (e.state where e.state= "w")
```

number would give you the number of extensions currently in the working state.

Avg (Average)

The function **avg** (average) computes the average of the values being queried. Average represents the sum of the items divided by the number of occurrences (count). As with count, you must set the expression that includes the average function to be equal to a variable such as x.

Sum

The function **sum** computes the total of values specified. **Atol** should most likely be used with this function since many of the field values of Manager IV are stored as strings; **sum** would operate on the converted data (after the string has been converted to an integer).

The following query will ask for the sum of extensions currently available.

Example

```
*target is "chicago"
*range of e is extn
*retrieve (total = sum (atol (e.extn)) where e.state = "a")
```

This query will give you a field called "total," which will be the sum of all working extensions. Functions such as **sum** must be assigned a variable (in this case total). This gives you the opportunity to assign meaningful field names to your queries.

FORMULATING QUERIES

Most likely, requests for queries will come from the Manager IV application administrators or their managers in the form of a request for information, and you will have to translate these into queries.

To do this, you will need to know:

- The target or subfile to be accessed
- The database files to be accessed
- The fields to be accessed
- The conditions when information should be printed (optional).

Some of this information will probably come to you with the request. If it doesn't, use query **help** to check on the relationship between the files and the fields that you are being asked to query. Briefly, to do this, enter the name of the file followed by the backslash go. To obtain this information about the extension file, enter **help \extn \g** and the fields of the file will be listed. For a list of Manager IV files, see Section 2 of this manual.

Sample Request

The telecommunications manager wants you to generate a report that lists the extension, set id and state fields for all working extensions of the LDN "9998833".

First, you must determine the target or subfile of the *database* that is to be searched. Because you have been given a specific LDN, that is easy to determine. The target is "9998833".

Next, we choose the *file* that is to be searched. We know that the extension, set id and state fields can all be found in the extension file. To check this, we can use the **CTM help** command. In this case, we would enter **help extn \g** to get a list of all the fields in the extension file. There is some additional information that is displayed such as the database key types, and the way the information is stored in the database. In the Manager IV database, most of the data is stored as strings, and the database keys are not relevant to Query, so those fields are omitted from the example on the next page.

Example

help extn \g

```
extn:
    #atbs (fields)
    34

    atb (field) name
    EXTN
    COS
    HUNT
    ANI
    CPG
    ATND
    CCG
    CCGMS
    NAMDB
    SETID
    STATE
    ACCID
    ...
```

The *fields* have already been determined for us, and the request was qualified to only those extensions that are working.

The Query

Once we have found the proper target, files and fields to search, it is easy to formulate the query. Once you have written the query, you need to enter the command **\g** to execute it.

Example

```
* target is "9998833"
* range of e is extn
* retrieve (e.extn, e.setid)
* where e.state = "w"
* \g
```

```
Query Results

EXTN   SET ID
12345   12345d
12555   23432
23445   NULL
```

As you can see from the example, only the fields that were specified within the parentheses are displayed as column heads. We could add the state field to the results of our query by adding that field to the **retrieve** statement. We could also re-edit the original query to remove the state field and replace that with a **sort** statement. In this case, we will sort on the state field.

Example

```
* target is "blue"
* range of e is set
* retrieve (e.extn, e.setid, e.state)
* sort by state
```

Query Results

```
EXTN   SETID  STATE
12555  23432  a
23548  23458  a
12345  12345  w
23443  23443  w
23546  23624  w
```

The **where** statement may not be limited to the state field. We could again rewrite the query using two range statements and changing the **where** statement to display results only when the field values of one file are the same as the field values of another file. For example, here is a slightly different version of the original query.

Example

```
*target is "9992233"
*range of e is extn
* range of s is setid
* retrieve (e.setid, s.setid, e.extn)
* where s.setid=e.setid
```

Query Results

```
SETID  SETID  EXTN
80000  80000  20019
80001  80001  20019
80002  80002  20019
80005  80005  20019
80000  20007  20019
80000  80000  20019
90000  90000  20019
90005  90005  20019
90008  90008  20003
```

When we are querying two different database files, as in the above example, we are really looking for the records where the two files overlap or are "joined." We are searching for information where the fields from one file are the same as the fields in the other.

TCM Examples

Here are some actual TCM examples. These examples will provide better experience with composing queries.

Request for Data from a Range of Extensions

You may want to find out some information about extensions within a specified range, for example, the class of service (cos), the names (namdb) assigned to those extensions and the current status of those extensions.

The *file* is the extension file, and the *fields* to be searched are the extension, class of service, name and state fields. Again, remember to check this information via **help** to ensure that these fields are in the file you are searching, in this case, the extension file.

Example

```
* target is "9992233"
* range of e is extn
* retrieve (e.extn, e.cos, e.namdb, e.state)
* where ((e.extn >"20008") and (e.extn < "20018"))
```

This request did not specify a particular status, or state, but did request that the information come from a particular group of extensions.

Query Results

```
EXTN   COS    NAMDB              STATE
20010  1      John Jones         w
20009  1      Stephen Smith      w
```

Request for Available Equipment Slots

The query language allows you fast access to information about the available slots for circuit packs and other equipment issues. This example searches for the available slot locations for a particular pack type. **CTM help** will display the fields assigned to the equipment file.

Example

```
help equip \g
equip:
  #atbs (fields)
   ----
  33 (number of fields)

  field name

  EQUIP
  LOC
  mod
  CAB
  MC
  CAR
  MCC
  SLOT
  CKT
  STLT
  PTYPE
  STATE
  DATE
  SRNUM
  ...
```

Pack type (ptype), slot and state are all fields that are found in the extension file. These are the fields that must be queried to get the information that is needed.

Example

```
* target is "9992233"
* range of e is equip
* retrieve (e.ptype, e.slot, e.state)
* where e.state = "a"
```

Query Results

```
PTYPE SLOT   STATE
NULL   00      a
sn270  NULL   a
sn270  NULL   a
sn270  NULL   a
sn270  NULL   a
sn270  NULL   a
sn270  NULL   a
sn270  08      a
```

The previous query can be rewritten to search for the available equipment locations for a particular pack type.

Example

```
* target is "9992233"
* range of e is equip
* retrieve (e.ptype,e.state, e.MOD, e.cab, e.slot)
* where e.ptype = "sn270"
```

Query Results

```
TYPE   STATE        MODULE      CABINET     SLOT
sn270  a            03          0           01
sn270  a            03          0           01
sn270  w            03          0           01
```

FM Examples

The query language can also be used to get reports on your trunks and other facilities tasks. Here are some real FM examples.

Class of Service of Trunk Groups

Class of service (cos) is assigned to every trunk and the FM administrator may want to know the cos of individual trunks. DID and ARS are two types of trunks and these are both also fields in the COS file.

Example

```
* target is "9994455"
* range of c is cos
* retrieve (c.did, c.ars)
* where c.did != " " and
     c.ars !=" "
```

Query Results

```
DID    ARS
1      1
3      3
```

Checking Feature Access Codes

The query language can be used to check the feature access codes that are in use in the various products. To find the field acronym for feature access code, use **help**.

Example

```
* target is "9994455"
* range of f is feadac
* retrieve (f.fdac)
```

Query Results

```
FDAC
#22
#33
```

Printing Queries

You can use the Core Query Processor (CQP) to execute saved query files. To do this, you must get into UNIX by quitting CTM (using the **quit** or **q** command). At the shell prompt, enter **cqp < filename.q | lp**; this will "pipe" (the vertical bar is called a pipe) the output of the query text to a printer.

Displaying Queries

You can display the results of queries on your screen using CQP also. Simply, enter **cqp <** *filename***.q**, (where *filename* is the name of the file where the query has been stored), without the pipe and the contents of your file will appear on the screen.

Saving Results of Queries

You can also use the Core Query Processor to save the results of queries or CTM help sessions to UNIX files.

Saving Help Sessions

You can use the CQP to save listings of the fields and files of the Manager IV database.

1. Enter **help** *filename* **\g** where *filename* is the name of a Manager IV file, for example, **help cos \g** would display all of the fields in the class of service file.
    - All the fields that are in that file will be displayed.

2. Enter **\w** *filename* to save the list that was generated. Choose whatever name you wish for *filename*.
    - If you look at that file, it will just display the CTM commands. It must be run through the CQP and then saved to another file before you can see the list.

3. Enter **cqp <** *filename* > *filename2* where *filename* is the name you saved the file as, and *filename2* is the name of the new file that is generated with this command.
    - The greater than (>) symbol is used to redirect a file to another file (*filename2*) after a process has been executed on the original file. You can use this file as you would any other UNIX file. It can be copied, printed, mailed or read into another file.

5. GENERATING REPORTS

Once you have developed a query, you may want to present that information in a format that best displays the material. The Manager IV Report Language gives you easy control over the appearance of the reports that are generated with the query language.

This section covers the steps involved in writing a report, the report language commands used to set up the basic structure of the report, the text formatting commands used to format reports, and the commands used to compile and run reports. This section also includes several examples of reports from TCM and FM.

CREATING REPORTS

This is an overview of the steps involved in creating a report using the Manager IV Report Language.

1. Create and test the query in the CTM environment.
2. Store the query text in a UNIX file. Remember to append the suffix **.q** to the filename that has been chosen, and then exit CTM.
3. Use a UNIX screen editor such as **vi** or **emacs** to create the report source code using the appropriate report language and text formatting commands.
4. Store the report source code as a UNIX file. Remember to append the suffix **.rpt** to the filename that has been chosen.
5. Use the vi command **read** to integrate the contents of the UNIX file containing the previously created query.
6. Use the Report Language command **.name** to specify the name of the UNIX file where the compiled report will be stored.
7. Save the completed report code including the query test to a UNIX file (filename must have an **.rpt** suffix).
8. Compile the contents of the file containing the report source code using the **sreport** command.
9. Execute the report using the **report** command. The filename used with the report command will be the same filename that has been specified in the **.name** command.

All of those steps will be explained in detail throughout this chapter.

REPORT LANGUAGE COMMANDS

There are several report language commands that can be used to set up the basic structure of your report. Some are required and others are optional. Each command is a macro statement and so is always preceded by a period.

Report Language commands are used with the text formatting commands to create reports. For example, **.header** is a report language command. It begins a block of text formatting commands, which set up the structure of the headings in the manner you wish. That block of commands ends with the next report language command.

Example

```
 .header
    .center
    .print "Name of Report"
    .tab +5
```

In the previous example, **.header** is a report language command, and the indented commands (**.center**, **.print**, **.tab**) are text formatting commands.

Name

name is a required statement that is used to name your report. This is the name that will be automatically given to the file that is created after the source code is compiled. The name you choose must be enclosed in quotation marks.

Example

```
.name "extset"
```

Query

query is also a required statement and directs the report program to use information gathered from the query text that you include. You can read in the query text that has already been developed and tested in CTM into the report code with the UNIX editor command **.read**.

Example

```
.query
    target is "2159233299"
    range of e is extn
    retrieve (e.extn, e.setid)
```

Sort

The **.sort** command indicates how to sort the data in the file before it is written to the report. Specify the name of the file to be sorted. The default for this command is ascending order.

Example

```
.sort extn
```

Example Results

```
10125
10128
10256
10456
```

To change the default and specify descending order:

```
 .sort extn:desc
```

Defvar

The **.defvar** command allows you to define a local variable, which is then set and incremented by the **.let** command.

The example for this command is included in the example for the **.let** command.

Let

This command sets the beginning value for a local variable, which is defined by the **.defvar** command. After setting the initial value, use **.let** again to increment the variable's value.

Example

```
.defvar numeric _count
.defvar string _pendstr
.header accid
        .let _count = 0
        .let _pendstr = "None pending"
.detail
        .if state = "p" .then
            .let _count = _count + 1
        .endif
.footer accid
        .newline 2
        .if _count > 0 .then
            .println count ("zzz,zzz")," pending"
        .else
            .println _pendstr
        .endif
```

Header

The **.header** command directs the report program to set up the top of your report, the top of each page, or the top of the column breaks according to the specifications that you select. This is an optional command. When used, **.header** initiates a block of text formatting commands.

Example

```
.header report
        .center
        .print "Manager IV Report"
        .nl 3
```

Text formatting commands are discussed in the next section, "Text Formatting Commands".

Footer

The **.footer** command directs the report program to set up the bottom of your report, the bottom of each page, or the bottom of the column breaks according to the specifications that you select. This is an optional command. When used, **.footer** initiates a block of text formatting commands.

Example

```
 .footer page
        .newline 2
        .left  .println  "page",page_number("-----")
```

Detail

The **.detail** command directs your report to set up the structure of each row of Manager IV database information in your report. This is a required command used with text formatting commands, which are discussed in the next section.

In the example below, the indented commands are text formatting commands.

Example

```
  .detail
        .print extn (c8)
        .print cos  (c 12)
```

Parameter Substitution

Parameter substitution allows you to make your queries and reports applicable to more than one sample. For example, rather than specifying a target as 9998833, you can make the target a variable by specifying it as *$target*.

Example

```
* target is "$target"
```

Then when you issue the report command from the shell prompt, you can specify the values for the variable parameters. For example, if you create a report named "extset" with the following query,

```
* target is "$target"
* range of e is extn
* retrieve e.extn, e.state, e.namdb
* where atol (e.extn) >= $lobnd
*    and atol (e.extn) <= $hibnd
```

you will issue the report command later from the shell prompt as follows:

```
report extset 9992233 0 150
```

If you forget to specify the parameters that you specified as variables (using the $ sign), you will be prompted immediately after you press **RETURN**.

For example, if you issue the following report command from the shell prompt,

```
report extset
```

you will be prompted as follows:

```
Enter value for ’target’:
Enter value for ’lobnd’:
Enter value for ’hibnd’:
```

NOTE: You must specify the variable parameters in the command line using the same order as they appear in your query.

TEXT FORMATTING COMMANDS

The following text formatting commands are used with the Report Language commands previously discussed. These are also macro statements and be preceded by a period.

We will describe the commands and then show how they are used in typical reports.

Center

The **.center** command centers text that follows a print statement in your report.

Example

```
.center
.print "A Sample Report"
```

In the example, the text "A Sample Report" is centered.

Format

This command sets up a default printing format for a column or set of columns.

Example

```
.format { {columnname } (format) }
```

where *columnname* is the name of a column or set of columns in the data being reported and *format* is a valid format specification.

Format Specifications

Be sure to use the right format for the type of data you are presenting:

- Character strings are specified using **C** format.
- **F** format specifies numeric data printed with or without a decimal point.
- **E** format specifies numeric data printed in scientific notation.
- **G** format specifies numeric data printed in either **F** or **E** format depending on what fits. This format also guarantees that decimal points align whether printed in **F** or **E** format.
- **N** format is the same as **G**, but decimal points do not necessarily align.
- **B** format is used for blanking out a field (for use with temporary formats in conjunction with the **.tformat** command.
- You may specify numeric data templates that define more complicated formats such as the inclusion of dollar signs or commas.
- You may specify date templates that define more flexible formats for date and time intervals.

A sign (+ or -) can precede each format specification to indicate right (+) or left (-) justification within the format field width. If no sign is given, the default is left justification for character fields and right justification for numeric fields.

Leftmargin

The **.leftmargin** command sets up the number of spaces in the left margin of each page of your report. To do this, follow the command with the number where you want the first column of text to begin.

Example

```
.leftmargin 10
```

Rightmargin

The **.rightmargin** command sets the right margin of each page of your reports. You specify the number of spaces next to the command.

Example

```
.rightmargin 80
```

Pagelength

The **.pagelength** command sets up the number of lines on each page of your report.

Example

```
.pagelength 45
```

Formfeeds

The **.formfeeds** command advances the page, which is useful when the report exceeds one page.

The .noformfeeds command suppresses the addition of formfeed characters to the end of each page.

Both commands can be used at either the top or bottom of each page. Make sure you specify these commands before any .header or .footer commands.

Example

```
.formfeeds
.footer
```

Left

The **.left** command left justifies text in your report.

Example

```
.print mod
.left
```

This example left justifies the values printed under the mod (module) column.

Lineend

Use this command to print the next text string at the end of the current output line (the first position after the last nonblank character on the line). This is useful in some advanced reports that use the **.tab** command extensively to move back and forth on the current output line.

Example

```
.tab 14 .pr "def  "
.tab 5 .pr "abc"
.lineend .pr ":xyz"
```

Need

Use the **.need** command to prevent a block of text from being interrupted by a page break. Put the **.need** command before the text block and specify the number of lines that the text block needs.

Example

```
.header page
       .need 2
       .print "Extension    Equipment    User"
       .print "Number       Location     Name"
```

Newline

The **.newline** or **.nl** command advances your text to the next line. You can specify the number of spaces in the command; the default (when there is no argument) is one.

Example

```
.print "Module"
.nl
.print "Carrier"
.nl 3
```

In this example, the word **Module** is printed, then **Carrier** is printed on the next line, and then three lines are advanced (two lines are left blank) before the next entry is printed.

Noformfeeds

The .noformfeeds command suppresses the addition of formfeed characters to the end of each page.

The **.formfeeds** command advances the page, which is useful when the report exceeds one page.

Both commands can be used at either the top or bottom of each page. Make sure you specify these commands before any .header or .footer commands.

Example

```
.noformfeeds
.footer
```

Nounderline

Use this command in conjunction with the **.underline** command to underline all text that you specify between these statements. The default underline character is a hyphen (-), which you can change by using the **.ulchar** command.

Example

```
.header page
      .need 3
      .ulchar "="
      .print "Extension    Equipment    User"
      .underline
      .print "Number       Location     Name"
      .nounderline
```

Print

The **.print** command allows you to specify text that should be included in your report, such as column headings.

Example

```
.print "Circuit"
.print "Carrier"
```

In this example, both words are printed.

Right

The **.right** command specifies the starting position of right-justified text in your report.

Example

```
.print "Module"
.right
```

In this example, the values placed under **Module** are right justified under that column.

Tab

The **.tab** command allows you to specify the starting position for text in your report.

```
.print "Carrier"
.tab +10
.print "Module"
```

In this example, **Carrier** is printed, and then **Module** is printed 10 spaces to the right.

You can combine these formatting commands with the report language commands to produce the kind of reports that you want.

Ulchar

When you underline text in your report, you have the option to specify the underline character. The default character is a hyphen (-). When you put the **.ulchar** command before the underline command, you can change the underline character to an equal sign (=), a letter, or any character.

The example for this command is included in the example for the

Underline

Use this command in conjunction with the **.nounderline** command to underline all text that you specify between these statements. The default underline character is a hyphen (-), which you can change by using the **.ulchar** command.

Example

```
.header page
        .need 3
        .ulchar "="
        .print "Extension    Equipment    User"
        .underline
        .print "Number       Location     Name"
        .nounderline
```

Control Flow

You can include the following control flow statements to specify different actions depending on certain conditions:

```
IF condition1  .THEN
             action1
ELSEIF condition2  .THEN
             action2
ELSE
             default_action
ENDIF
```

Example

```
.detail
        .print extn .tab +2
        .if state = "w" .then
            .print "working"
        .elseif state = "a" .then
            .print "available"
        .else
            .print "pending"
        .endif
```

SAMPLE REPORT CODE

The first example is based on the query that we discussed earlier in Section 4. This query produces a list of the extension numbers and set ids of all working extensions. As a query, this file was called **extset.q** and as report code it is called **extset.rpt**. It is set up to produce a simple three column format.

```
.name "extset"


.query
     target is "2015556868"
     range of e is extn
     retrieve (e.extn, e.setid, e.state)
     where e.state = "w"

.sort extn

.leftmargin 1

.rightmargin 80

.formfeeds

.header report
       .nl 10
           .center
           .print "ADMINISTRATOR DEVELOPED"
           .nl
           .center
           .print "EXTN/SETID REPORT"
           .nl
           .center
           .print current_day, "    "
           .newpage 1


.header page
           .nl
           .print "EXTN"
           .tab +10
           .print "SETID"
           .tab +10
           .print "STATE"
           .tab +10
           .nl
```

```
 .detail
         .print extn
         .tab +10
         .print setid
         .tab +10
         .print state
         .tab +10
         .nl
```

As you look at this example, you can see how the various formatting commands are used with the Report Language commands. Each command is preceded by a period, and you can indent the code or not as you wish.

Compiling Reports

Once you have written this report source code, you must compile it before you can see the results. To do that, simply enter the command **sreport extset.rpt**. You will see the following message displayed on your screen:

```
"AT&T Manager IV REPORT COMPILER--compiling file 'extset.rpt'
Start of specifications for report 'extset'
Writing specifications for report 'extset'"
```

The file extset was created when you compiled the report. At this point your directory should include the following three entries:

```
"extset.q is the query
extset.rpt is the source code for the report
extset is the compiled version of extset.rpt"
```

Running Reports

Now that you have compiled the report, you can display it on your screen, save it to a file, or print a hardcopy version.

Displaying Reports

To display the report on the screen, use the command **compiled version of the report code. That name comes from the filename assigned in the report code with the name** command.

For the example that we have been using, that command is **report extset**.

Saving Reports

To save the report as a UNIX file, use this command **report -f fileout filein** where *filein* is the name of the compiled report and *fileout* is the name you choose for the UNIX file.

For this example, if we call the new file **set**, the command is **report set extset**. **Set** will contain the same contents as the file **extset** and exist in the same directory. You can use it as you would another UNIX file.

SAMPLE REPORT

After **exttset.rpt** has been compiled, running the report **extset** results as follows:

```
                ADMINISTRATOR DEVELOPED
                     EXTN/SETID REPORT
                          Fri

EXTN     SETID      STATE
00054              w
00055              w
00056              w
00057              w
00060              w
00061              w
00066              w
00068              w
00109              w
02001              w
02002    02000A      w
02003              w
20000              w
20001              w
20002              w
20003              w
20004              w
20005              w
20006              w
20007    20007      w
20008    20008      w
20009              w
20010              w
20018              w
20019    80000      w
21000              w
```

This is only one example of a report generated with the Manager IV Query and Report Languages. This section will contain several examples that you might use.

TCM Reports

This is an example of a report listing available equipment locations.

Sample Report 1

```
.name "ptypeseqs"
.query
     target is "$target"
     range of e is equip
     retrieve (e.ptype,e.state, e.mod, e.cab, e.slot)
     where e.ptype = "$ptype"


.header report
     .nl 2
     .center
     .print "Equipment Locations"
     .nl 3
     .left ptype
     .print "TYPE"
     .left state
     .print "STATE"
     .left mod
     .print "MODULE"
     .left cab
     .print "CABINET"
     .left slot
     .print "SLOT"
     .nl

.detail
      .print ptype (c12), state (c8), mod (c12),
             cab (c12), slot (c8)
      .nl
```

This query has been made generic by making both target and pack type into variables whose values are entered at run-time.

For this example, you are prompted:

```
Enter value for target:          target:
Enter value for ptype:            ptype:
```

For this example, **9992222** was entered for the target, and **sn270** was entered for the pack type. The results of this report (after compiling) are:

```
             Equipment Locations


TYPE        STATE   MODULE      CABINET     SLOT
sn270       a       03          0           01
sn270       a       03          0           01
sn270       w       03          0           01
```

Sample Report 2

This next report looks for information from a specific range of extensions.

```
.name phones
 .query
    target is "9992233"
    range of e is extn
    retrieve (e.extn, e.cos, e.namdb, e.state)
    where ((e.extn >"20008") and (e.extn < "20018"))

.header page
    .print "EXTN        COS        NAME           STATE"
    .nl 2
.detail
   .print extn
   .tab +5
   .print cos
   .tab +5
   .print namdb
   .tab +10
   .print state
   .tab +5
   .nl 2
```

The results of this report are:

```
EXTN        COS        NAME        STATE

20010             1          peter martin      w

20009             1          stephen jones  w
```

6. CTM AND QUERY LANGUAGE COMMAND DIRECTORY

This section includes commands for both the Core Terminal Monitor (CTM) and the Query Language. The commands are combined into one section because queries are developed and executed with CTM commands.

Each command is listed separately and described as a CTM or query command. To help you find information quickly, each entry in this command directory includes the following information:

- Description of the command
- Syntax
- Example
- Sample output (where appropriate)

EDIT (E)

The **edit** or **e** command is a CTM command that is used to edit the current contents of the query buffer. A UNIX editor such as **vi** or **emacs** is used.

Syntax

e or **edit**

Example

edit will read the contents of the query buffer into a vi file.

GO (G)

The **go** or **g** command is a CTM command that is used to execute the query. If you are executing a query that is in the query buffer, just follow the query with the **go** command or type **go** *filename* (where *filename* is the name of the file to be read in) to execute a query that is not currently in the buffer.

Syntax

g or **go**

Example

```
* target is "9992233"
* range of a is aar
* retrieve (a.aarky, a.srnum)
* where a.aarky=0001
* \go
```

The message " * * executing" will be displayed followed by the results of your query.

HELP (H)

The **help** or **h** command is used to list available CTM commands. When combined with a **go** command, it is used to print a list of the files and the fields that are in your Manager IV database.

Syntax

To display a list of all CTM commands:
* **help** or \h

To display a list of all the Manager IV database files:
* **help \g**

To display a list of the fields in a particular file:
***help** *filename* **\g** (where *filename* is the name of
the particular database file that you want field information from)

Examples

To list all CTM commands:

* **help**

Example Output

```
append (a)
print  (p)
write  (w)
 ...
```

Example

To get a list of all the files in the Manager IV database:

* **help \g**

Example Output

```
EXTN

relnm    #atbs    #pgs     pck'd
COS
EQUIP
TRKGP
AAR
ARS
 ...
```

relnm represents the relation name or the name of the file, and **#atbs** stands for the number of attributes (fields).

Example

To get a list of all the fields in the extension (extn) file.

*** help extn \g**

Example Output

```
extn:
atbs      #pgs      pck'd
34        998700 y

atb name           type      length    key
EXTN    string
COS
HUNT
ANI
CPG
ATND
CG
CCGMS
 ...
```

INCLUDE(I)

Include is a CTM command that reads a file into the query buffer. After the **include** command has been executed, use the CTM commands **print** or **edit** to display the file.

Syntax

\i *filename*
(where *filename* is the name of the file to be read into the query buffer)

Example

\i equipment.q

PRINT(P)

Print is a CTM command that displays the current contents of the query buffer to the screen.

Syntax

\p

Example

```
*
*  target is "blue"  (current contents of the query buffer)
*  range of e is equip
*  retrieve (e.slot)
*  where e.ptype = sn270
*  continue
*
```

At this point, you may execute this query with the **go** command. You will see the message "executing...", and then the results of the query will be displayed.

QUIT (Q)

Quit or **q** is a CTM command that allows you to exit the CTM environment and get back into the UNIX shell. The command **write** must be used with **quit** to save a copy of a file. If **\quit** is used without the **write** command, the file, or changes to the file since the last **write** command was issued, will not be saved.

Syntax

\q

Example

```
* target is "9997575"
* range of e is extn
* retrieve (e.setid, e.namdb, e.state)
* where e.state = "a"
* \w extension.q
* \q
$  (the UNIX shell prompt)
```

This example traces one query, which will be called **extension.q**, to the quit statement.

RANGE

Range is a required query statement that identifies the particular database file to be searched. There can be more than one range statement specified and more than one database file searched in a query.

Syntax (one target)

* range of *f* is *filename*
(where *f* is the identifier and *filename* is the name of the Manager IV database file being searched)

Syntax (two targets)

* range of *f* is *filename* in *t*
(where *f* is the identifier that describes which Manager IV file is being used, *filename* is the name of the Manager IV file itself, and *t* is the name of the identifier given to the target being accessed)

Example (with one target)

```
* target is "blue"
* range of e is extn
* range of eq is equip
```

Example (two targets)

```
* target t is "blue"
* target p is "pegasus"
* range of a is acct in t
* range of e is extn in p
```

RESET (R)

The CTM command **reset** is used to erase the contents of the query buffer. This command should be used if there are too many mistakes in the original query to edit it, or there is no need for the query that is in the query buffer.

Syntax

\r

Example

This example demonstrates one usage for the **reset** command. After the query was completed, there was no longer any reason to keep it.

```
* target is "9993344"
* range of e is extn
* retrieve (e.setid, e.namdb)
* where e.setid >"23416" and
            e.setid < "24000"
*\r  (the query buffer is now empty)
```

RETRIEVE

Retrieve is a required query statement that identifies the Manager IV fields to be viewed. Several fields may be specified in this statement including fields from one or more database files. Information from the fields specified in the parentheses will be displayed.

All the fields in a particular file can be viewed by using the field name all, but this is *not* recommended because such a request may take very long to process.

Syntax

*** retrieve (f.***fieldname1*, f.*fieldname2*, and fi.*filename1*)
(where *fieldname*1 and 2 are the names of the fields of the Manager IV files that are being searched)

Example

```
* range of e is extn
* range of eq is equip
* retrieve (e.setid, e.state, eq.ckt)
```

SORT

Sort is an optional query language command that can be used to sort the value of the fields in the results of the query. The default used with the **sort** command is ascending. To change the default and specify descending, append **:d** to the sort command.

Syntax (using the default)

sort by *field*
(where *field* is the name of the field to be sorted)

Example

```
* target is "green"
* range of e is extn
* retrieve (e.extn, e.setid, e.state)
* sort by setid
```

Example Output

```
EXTN   SETID  STATE
5043   1234   w
5056   1235   r
6000   1280   w
```

If *descending* order were desired, the query would be changed as follows:

```
* target is "green"
* range of e is extn
* retrieve (e.extn, e.setid, e.state)
* sort by setid:d
```

TARGET

Target is a required statement in a query that identifies the particular Manager IV database that is to be searched. A database is usually identified by a Listed Directory Number (LDN) or PBX identifier, usually an alphanumeric string such as "zeus."

More than one target can be searched at a time.

Syntax (one target)

```
* target is "number"
```

Example (one target)

```
* target is "blue"
```

Syntax (Two targets)

```
* target t is "blue"
* target p is "(201)123-4567"
```

Example (Two targets)

```
* target t is "blue"
* target p is "1234567"
* range of a is acct in t
* range of e is extn in p
* retrieve (e.accid, a.cpgrp, e.extn)
* where (a.accid=e.accid)
```

In the next example, target is used as a variable ($target), which can be included in the report code's query. It will prompt you to select the target to be searched before the report is executed.   The use of this variable allows you to develop generic queries and then embed them in report source text.

```
* target is "$target"
```

WHERE

The optional query statement **where** allows you to restrict the search to conditions that you specify. This statement handles logical operators and can be as complex as you wish.

The logical operators supported are:

equals (same as)	=
not equal	!=
greater than	>
greater than or equal to	>=
less than	<
less than or equal to	<=
and	
or	

Syntax

* where *i.fieldname* = value
(where *i.fieldname* is the name of the field name being qualified, and *i* is the identifier that indicates from which database file the field name is drawn)

Examples

```
* where e.state = "w"

* target is "blue"
* range of e is extn
* range of c is cos
* retrieve (e.cos, c.cos)
* where e.cos = c.cos
```

WRITE (W)

The **write** command is used in both CTM and UNIX editors (**vi** or **emacs**) to store a query in a file. Follow the **\write** command with the name of the file you choose. To distinguish between queries and reports in a crowded directory, use the suffix **.q** after the filename to mark it as a query.

Syntax

\w *filename*.q
(where *filename* is the name chosen as the UNIX file name)

Example

In this example, the query will be saved to the filename **equipment.q**.

```
* target is "9993344"
* range of e is equip
* retrieve (e.slot, e.car)
* where e.ptype = "sn240"
* \w equipment.q  (equipment is the name of the UNIX filename)
```

7. REPORT LANGUAGE COMMAND DIRECTORY

This section includes commands used in the Manager IV Report Language. This includes the Report Language commands, the text formatting commands that are used with Report Language commands, and the commands used to compile and run reports generated with the Query and Report Languages.

Each command is listed separately and described as a text formatting, or report language command. To help you find information quickly, each entry in this command directory includes the following information:

- Description of the command
- Syntax
- Example
- Example Output (where appropriate)

All commands in this section excluding **report** and **sreport** *must* be preceded by a period (.).

CENTER

The **center** command centers the next block of text specified in the print command.   All leading and trailing blanks are removed from the text before it is placed in the output line. This command is used in conjunction with one of the report language commands.

Syntax

.report command (such as .detail or .header)

```
   .center
   .print "Text to be centered"
```

Example

```
 .header report
    .center
    .print "Pack Information"
```

Example Output

```
                              Pack Information
```

CONTROL FLOW

You can include the following control flow statements to specify different actions depending on certain conditions:

- IF
- THEN
- ELSEIF
- ELSE
- ENDIF

Syntax

```
IF condition1  .THEN
            action1
ELSEIF condition2  .THEN
            action2
ELSE
            default_action
ENDIF
```

Example

```
.detail
      .print extn .tab +2
      .if state = "w" .then
          .print "working"
      .elseif state = "a" .then
          .print "available"
      .else
          .print "pending"
      .endif
```

CURRENT_DATE

Current_date is a text formatting command that is used to specify the current date on the report. This is particularly useful if you are going to set up report templates. This command enables you to get the correct date without having to enter it into the report.

Syntax

.print current_date, " "

Example

```
.center
.print current_date, " "
```

Example Output

This example prints the current date on the report and centers it.

DEFVAR

The **.defvar** command allows you to define a local variable, which is then set and incremented by the **.let** command.

must begin with "_"
numeric string only valid types
can have up to 16 local variables

Syntax

.defvar numeric _count

Example

```
.defvar numeric _count
.defvar string _pendstr
.header accid
        .let _count = 0
        .let _pendstr = "None pending"
.detail
        .if state = "p" .then
           .let _count = _count + 1
        .endif
.footer accid
        .newline 2
        .if _count > 0 .then
           .println count ("zzz,zzz")," pending"
        .else
           .println _pendstr
        .endif
```

DETAIL

Detail is a required statement that directs your report program to set up the structure of each column of Manager IV database information in your report. It is used with report formatting commands to set up the spacing between columns.

Syntax

Each report language command is preceded by a period.

.detail

Example

This example includes the query and detail sections from a report. It does not include any other sections.

```
. query
   target is "9993344"
   range of e is equipment
   retrieve ( e.type, e.state, e.mod, e.car, e.cab)
   where e.state = "a"

.detail
   .print ptype
   .print state  .tab 7
   .print mod    .tab 12
   .print cab    .tab 18
   .newline
```

Example Output

```
sn270   a     03    0
sn270   a     03    0
sn270   a     03    0
```

ELSE, ELSEIF, ENDIF

See "Control Flow" earlier in this section.

FOOTER

The **footer** command is an optional command that sets up the bottom of your report, the bottom of each page, or the bottom of the column breaks. It identifies the start of a block of formatting commands that are executed at the end of a page or the end of the report. All commands between one **.footer** command and any subsequent **.header**, **.footer** or **.detail** command are considered to be part of the first footer action.

The **.footer** command should follow the opening report setup commands.

Syntax

.footer *type*
(where type can be either report, page or column name)

Example

This is an example of a footer block that defines the footers for a page.

```
.footer page
    .nl 2
    .left
    .print "page" , "page_number ("------")
    .nl
```

Example Output

page 1

FORMAT

This command sets up a default printing format for a column or set of columns.

Syntax

.format { {*columnname* } (format) }

where *columnname* is the name of a column or set of columns in the data being reported and *format* is a valid format specification.

Format Specifications

Be sure to use the right format for the type of data you are presenting:

- Character strings are specified using **C** format.
- **F** format specifies numeric data printed with or without a decimal point.
- **E** format specifies numeric data printed in scientific notation.
- **G** format specifies numeric data printed in either **F** or **E** format depending on what fits. This format also guarantees that decimal points align whether printed in **F** or **E** format.
- **N** format is the same as **G**, but decimal points do not necessarily align.
- **B** format is used for blanking out a field (for use with temporary formats in conjunction with the **.tformat** command).
- You may specify numeric data templates that define more complicated formats such as the inclusion of dollar signs or commas.
- You may specify date templates that define more flexible formats for dates and time intervals.

A sign (+ or -) can precede each format specification to indicate right (+) or left (-) justification within the format field width. If no sign is given, the default is left justification for character fields and right justification for numeric fields.

FORMFEEDS

The **formfeeds** command advances the page, which is useful when the report exceeds one page.

The **noformfeeds** command suppresses the addition of formfeed characters to the end of each page.

Both commands can be used at either the top or bottom of each page. Make sure you specify these commands before any .header or .footer commands.

Syntax

.formfeeds

Example

```
.formfeeds
.footer
```

Example Output

This example results in formfeeds occurring before the footer is printed.

HEADER

The **header** command sets up the top of your report, the top of each page, or the top of the columns. The header command is used with report formatting statements to set up the headers as you wish. All commands between one **.header** command and any subsequent **.header**, **.footer**, or **.detail** command are considered to be part of the first header action.

The **.header** command should follow any overall report setup commands, including:

- pagelength
- rightmargin
- leftmargin
- format
- position
- width

Syntax

.header *type*
(where *type* indicates whether the
header is for report, page or column)

Example

This example includes the query and header specifications.

```
.query
     target is "9994455"
     range of e is equip
     retrieve (e.state, e.mod, e.car, e.ckt)
     where e.state = "w"

.header report
    .newline 2
    .center
    .print " EQUIP LOCS AVAIL"
    .nl 2
    .tab 1
    .print "State"
    .tab 8
    .print "Module"
    .tab 14
    .print "Carrier"
    .tab 22
    .print "Circuit"
    .nl
```

Example Output

```
                         EQUIP LOCS AVAIL

State    Module Carrier  Circuit
```

IF

See "Control Flow" earlier in this section.

LEFT

The **left** command left-justifies the next block of text a specified position relative to the last output, absolute or else to a default position for a column. The position is moved *n* positions relative to the last output of text, when it is signed (+ or -). All leading and trailing blanks are removed from the text before it is placed in the output line.

Syntax

.left *fieldname*
(where *fieldname* is the field whose value should be left-justified)

(where *nlines*, if signed, is moved *n* spaces relative to the last position output. If the *n* is unsigned, the number indicates the absolute position of the text.)

.left [[+ | -] n | *columnname*
(where *n* is the position to which the next text output is justified.)

Example

```
 .header
     .print "State"
     .tab 5
     .print "Module"

 .detail
     .left
     .print "state"
     .tab 5
     .left
     .print "module"
     .nl
```

This results in the following output with the field values left justified under state and module.

Example Output

```
State    Module
w        0002
a        0003
```

LEFTMARGIN

Leftmargin is an optional report language command that sets up the number of spaces in the left margin of each page of your report. The command is followed by the number you have chosen for your margin.

Syntax

.leftmargin *number*

Example

```
.leftmargin 1
```

Example Output

This example results in the margin being indented one space.

LET

This command sets the beginning value for a local variable, which is defined by the **.defvar** command. After setting the initial value, use **.let** again to increment the variable’s value.

Syntax

.let _count = 0
.let _pendstr = "None pending"

Example

```
.defvar numeric _count
.defvar string _pendstr
.header accid
        .let _count = 0
        .let _pendstr = "None pending"
.detail
        .if state = "p" .then
            .let _count = _count + 1
        .endif
.footer accid
        .newline 2
        .if _count > 0 .then
            .println count ("zzz,zzz")," pending"
        .else
            .println _pendstr
        .endif
```

LINEEND

Use this command to print the next text string at the end of the current output line (the first position after the last nonblank character on the line). This is useful in some advanced reports that use the **tab** command extensively to move back and forth on the current output line.

Syntax

.lineend .pr ":xyz"

Example

```
.tab 14 .pr "def"
.tab 5 .pr "abc"
.lineend .pr ":xyz"
```

Example Output

```
abc     def:xyz
```

NAME

Name is a required report language command, which must be the first command specified for a report. All commands following the name command are considered to be a part of that report. The name chosen for the report does not have to be the name of the query, but can be any name chosen for this report. This name will be used when compiling the report.

Syntax

.name *filename*
(where *filename* is the name of the report)

Example

```
.name extset
```

Example Output

After the report has been compiled, **extset** will automatically appear as a file in the current directory.

NEED

Use the **need** command to prevent a block of text from being interrupted by a page break. Put the **need** command before the text block and specify the number of lines that the text block needs.

Syntax

.need 2

Example

```
.header page
        .need 2
        .print "Extension    Equipment    User"
        .print "Number       Location     Name"
```

Example Output

This example allows the following header to be kept on one page without a page break interruption:

```
Extension   Equipment   User
Number      Location    Name
```

NEWLINE

The **newline** command must be specified to advance to a new line. The **newline** command is followed by the number of lines to be advanced. If no number is specified after the command, only one line is advanced. **Newline** is a formatting command that is used in conjunction with the report language commands. It can be either written out in full or abbreviated to **nl**.

Syntax

.nl *nlines*
(where *nlines* is the number of lines to be skipped)

One is the default for **nlines**.

Example

```
.header
   .center
   .print "First Report"
   .nl 3
   .print "State"
   .nl
   .print "Module"
```

Example Output

```
                              First Report



State

Module
```

NEWPAGE

The **newpage** command initiates an immediate page break with an optional change in the page number. If no page number is specified, the next page will be incremented by one as usual. However, you can specify a page number in this command and it will skip a page and print whatever page number you specified.

Syntax

.newpage
(this command will skip a page and print the page number as usual)

.newpage *pagenumber*
(where *pagenumber* is the number you wanted printed
after the page break)

Example 1

```
.newpage
```

Example Output

In this example, a page is skipped and the page number increments as usual.

Example 2

```
.newpage 25
```

Example Output

In this example, a page is skipped and the next page number is 25.

NOFORMFEEDS

The noformfeeds command suppresses the addition of formfeed characters to the end of each page.

The **formfeeds** command advances the page, which is useful when the report exceeds one page.

Both commands can be used at either the top or bottom of each page. Make sure you specify these commands before any .header or .footer commands.

Syntax

.noformfeeds

Example

```
.noformfeeds
.footer
```

Example Output

This example suppresses formfeeds before printing the footer.

NOUNDERLINE

Use this command in conjunction with the **underline** command. When you use the underline command, a line is placed under specific characters from where the **underline** command begins. The **nounderline** command turns-off the **underline** command preventing any further text from being underlined.

Syntax

.nounderline

Example

```
.header page
        .need 3
        .ulchar "="
        .print "Extension    Equipment    User"
        .underline
        .print "Number       Location     Name"
        .nounderline
```

Example Output

```
Extension    Equipment    User
Number       Location     Name
=====================
```

PAGELENGTH

The **pagelength** command controls where page breaks occur. You specify the pagelength command at the start of your report specification before any header or footer commands are specified. The default is 61 lines per page if the report is written to a file, and 23 lines per page if the report is written to a terminal.

Syntax

.pl *nlines*
(where *nlines* is the number of lines per page)

Example

.pl 24 (24 lines per page)

Example Output

This example results in each page having only 24 lines of output.

PRINT

The **print** command is a text formatting command that is used to specify text to be printed in your report. By preceding the **.print** command with the positioning commands such as **.newline** or **.tab**, you can adjust the printout's appearance.

Syntax

.print "*heading" fieldname*
(where *heading* is the name of the field heading, and fieldname is the field whose value is to be printed. Either of these fields can be omitted.)

Example

```
.header page
    .print "STATE"
    .tab 5
    .print "MODULE"
    .tab 5

 .detail
    .print state
    .tab 5
    .print mod
    .tab 5
```

Example Output

```
STATE  MODULE
a        234
r        265
```

QUERY

Query is a required report language command that indicates the beginning of the query code that creates the data to be reported. This query follows the same rules as all other queries. You can use as many lines as you need to specify the query. The **-sreport** command detects the end of the query by the start of a new report formatting command.

Syntax and Example

```
.query
    target is "8342222"
    range of e is extn
    retrieve (e.extn, e.cos,e.namdb)
    where ((e.extn > "20008") and (e.extn < "20018"))
```

REPORT

The **report** command is used to run reports. There are two variations of this command: one to display the report to the terminal screen, the other to save the report as a UNIX file.

Syntax

To display a report to the screen:
report *filename*
(where *filename* is the name of the compiled report)

To save the report to a UNIX file:
report -f *fileout filein*
(where *fileout* is the name of the UNIX file to be
generated and *filein* is the name of the compiled report)

Example

In the examples below, **extset** is the name of the compiled report and **extset.unix** is the name of the UNIX file.

To display the report to the screen:
report extset

To save the report to a file:
report -f extset.unix extset

RIGHT

The **right** command is a text formatting command that right justifies the text printed in the next **print** command to a specified position. When *n* is signed (+ or -), the position is moved *n* positions relative to the last position output. Unsigned, the position is the absolute position in the output line. The default value is the right margin of the text. All leading and trailing blanks are removed from the text before being placed in the output line. It is used in conjunction with a report language command.

Syntax

.right *fieldname*
(where fieldname is the name of the field whose value should be right-justified)

.right [[+ | -] n | *columnname]*
(where *n* is the position to which the next block of text is right justified, and *columnname* refers to the name of the column being reported.

Example

```
  .header page
      .print "State"
      .tab 5
      .print "Module"

   .detail
     .right state
     .right mod
```

Example Output

```
State   Module
    w       00
    a       03
```

RIGHTMARGIN

Rightmargin is an optional command that sets up the number of spaces in the right margin of each page of your report.

Syntax

.rightmargin *number*
(where *number* is the number of spaces in the right margin)

Example

.rightmargin 80

Example Output

This example results in the right margin being indented 80 spaces.

SREPORT

Sreport is the command used to compile report language source code. The compiled report takes as its name the filename found after the **name** command in the original report source code.

Syntax

sreport *filename.rpt*
(where *filename.rpt* is the name of the report
source code to be compiled)

Example

sreport extset.rpt (extset.rpt is the name of the report to be compiled)

SORT

Sort is an optional command that specifies the order of the results of your report program and allows the report to be presented in the order that you select: descending (d) or ascending (a).   The default order is ascending.

Syntax

.sort *fieldname* order
(where fieldname is the name of the field to be sorted)

Example

This example begins with the query and then specifies what field to sort on.

```
.query
    target is "9993344"
    range of e is extn
    retrieve (e.extn, e.state)
    where ((e.state = "w") and (e.extn < "20008"))
```

.sort extn: desc

Example Output

```
EXTN              STATE
20005             w
20001             w
20000             w
```

TAB

The **tab** command is a text formatting command that specifies the position where the next text will be printed. **Tab** can be followed by a number. If signed (+ or -), the number represents a relative change from the last position output. If unsigned, the number represents an absolute position in the line. **Tab** is used in conjunction with the report language commands.

Syntax

.tab *n*
(where *n*, if signed, represents a relative change from the last position and, if unsigned, represents an absolute position in the line)

Example

```
.detail
   .print "PBXID:",
   .tab 7
   .print "PORT:",
```

Example Output

```
PBXID:          PORT:
```

THEN

See "Control Flow" earlier in this section.

ULCHAR

When you underline text in your report, you have the option to specify the underline character. The default character is a hyphen (-). When you put the **ulchar** command before the underline command, you can change the underline character to an equal sign (=), a letter, or any character.

Syntax

.ulchar "="

Example

```
.header page
        .need 3
        .ulchar "="
        .print "Extension    Equipment    User"
        .underline
        .print "Number       Location     Name"
        .nounderline
```

Example Output

```
Extension   Equipment   User
Number      Location    Name
=====================
```

UNDERLINE

Underline is a text formatting command that is used to underline a word, or line, or block of text. This could be used for headings, or to emphasize important parts of the report. Underlining is initiated with the **underline** command and ended by the **nounderline** command.

Syntax

```
 .underline
      .report writer commands
      .nounderline
```

Example

```
  .header report
            .underline
            .print "TCM REPORT"
            .nl 2
            .print "Module 1"
            .nounderline
```

Example Output

TCM

Module 1

INDEX