The ADMINS Manual

www.admins.com

ADMINS for Windows
Version 8.4 March 2012

Chapter 1:Introduction

ADMINS consists of a number of related executable programs ("commands") that together form an integrated environment for the development and operation of administrative and management information applications. The ADMINS commands should be located in a folder that is part of the users PATH environment variable. This folder is usually identified by the logical name[1] ADM$DIST.

At the most basic level, to execute an ADMINS command, you can type its name at the system prompt. The ADMINS command then either reads instructions typed by the user or reads an instruction file specified by the user. You can also click on the command's name or icon in a folder listing, or click on a "shortcut" icon for it, or click on the name or icon for an item that has been associated with the command.

The ADMINS Data Dictionary, described in Appendix I: "ADD: The ADMINS Data Dictionary" is a repository for information about the elements that make up an ADMINS-based information system. Using the ADMINS Data Dictionary, while not required, provides an enhanced capability for development, maintenance and documentation of complex ADMINS applications.

The Data Dictionary, the data management tools, the on-line data entry update and query tools, the reporting tools, the file processing commands for sorting, moving, and calculating, the relational product command, and the analysis tools - are all parts of one integrated system design.

On-line messages, displays, debuggers and test modes-of-operation are used to make visible the data operations being performed, and to help developers and users see how the ADMINS operations are affecting the data on a step-by-step basis.

ADMINS can be used in a variety of hardware and software environments ranging from single-user workstations to networks of mainframe-class processors that support entire organizations. ADMINS syntax and application source code is very portable across these myriad alternatives, enabling developers to create applications that transfer easily to new environments and run efficiently once there.

1. See Appendix C.4 "ADMINS Logical Names for Win32" for a general discussion of logical names. See Appendix B: "Special Logical Names used by ADMINS" for a list of the logical names used to configure ADMINS.

1.1 Using ADMINS

ADMINS provides a wide range of facilities. The beginner should understand the fundamental concepts of ADMINS before pursuing the more advanced facilities.

The major commands in ADMINS are as follows:

Command Name	Description
AdmDefine	The DEFINE command is used to create a file on a disk. The file is organized and formatted to store records of a particular structure, as described in the file definition instruction text.
AdmMove	The MOVE command is used to move records from one file to another.
AdmSort	The SORT command is used to sort records from one file to another.
AdmScreen	The SCREEN command is used to compile, i.e. prepare and organize, screen descriptions for use by the transaction processor.
AdmTrans	The TRANS command is used to enter, update, and/or display data on the video terminal under the control of the screen description compiled by the SCREEN command.
AdmReport	The REPORT command is used to prepare printed reports.
AdmCmp	The CMP command is used to compile record maintenance procedures which may be used with TRANS, MAINT, MOVE, PROD and REPORT. Record maintenance procedures contain programming logic to calculate, control, and otherwise effect record processing.
AdmMaint	The MAINT command performs a record maintenance procedure on a file, on a record by record basis.
AdmProd	The PROD command is used for the transfer and manipulation of data from records in one file linked to records in another file based on the key value relationships between the files.
AdmCom	The AdmCom command executes a sequence of other ADMINS commands to perform an application function, e.g. run a payroll.

There are many other facilities in ADMINS that are covered in this Manual. Once the fundamental concepts of ADMINS are understood, you will see how the advanced facilities may be applied to specific application problems.

1.2 ADMINS Manual

The ADMINS Manual provides a complete reference to all of the ADMINS commands and facilities, including in each case the purpose and syntax of the command, and detailed examples. Generally, each section of the manual corresponds to an ADMINS command or major facility.

This Manual is intended as both a guide to the purpose and function of ADMINS commands for the beginning or casual ADMINS user, and also as a complete reference guide to **all** the specific and detailed dialogue and syntax as required by the experienced ADMINS user.

1.2.1 Manual Conventions

The ongoing text of this Manual is presented in the print font and style used in this paragraph. Text may be emphasized using **bolding** or UPPERCASE.

In examples, the characters displayed or printed by ADMINS and the operating system are shown as they appear on hard copy or on a video screen. The characters which are typed responses by the user to ADMINS and operating system prompts are shown in lowercase. For example:

```
$ move
Input file....: telfon.mas
Output file...: newtel.mas
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
16:10:20.68
************************************************************
875 records moved, total 875 records in N2.MAS
16:10:26.52
$
```

Syntax descriptions and examples may contain characters which have a specific meaning. These are as follows:

Character	Description
CR	The characters "CR" mean "carriage return" and indicates that the user simply presses the RETURN or ENTER key. This is illustrated in the above example.
[ ]	Square brackets, "[ ]", indicate that the enclosed item is optional. (Square brackets are not, however, optional in the syntax of a directory name in a file specification.)
...	Horizontal ellipsis indicates that additional information may be included on a line, or that not all of the lines of an example are included.

1.3 ADMINS Instruction Files

ADMINS commands are either entirely interactive, i.e. they receive all their instructions as direct responses from the terminal, or ADMINS commands may utilize an instruction file which has been prepared in advance. The AdmDefine, AdmScreen, AdmReport, and AdmCmp commands, among others, use an instruction file. AdmTrans, AdmAded, AdmMove, and AdmSort, among others, can operate directly on-line. As we shall see, **all commands** (except AdmTrans) can operate from a "higher level" instruction file called an ADMINS command file. Also, using advanced techniques such as those described in Appendix H.13.14 “SETKEY - Simulate Keystrokes in TRANS” of this Manual, even TRANS can receive its keystrokes from sources other than the video display terminal.

Instruction files are prepared using a text editor. While the choice of which text editor to use is left to the user, the ability to use a text editor to create and modify instruction files is a prerequisite to using ADMINS to develop and maintain applications.

The instruction file for a specific command is described in detail in the section for the command. However, there are certain general conventions which apply to all instruction files.

1.3.1 Comments

Lines in ADMINS instruction files that begin with an asterisk (*) are treated as comments and are not processed. Various ADMINS commands also support additional methods of adding comments to instruction files. File definitions (.DEF files) may include comments at the end of field description lines by enclosing the comment in quotation marks (").

The exclamation point (!) can also be used in instruction files as a comment delimiter, but not in the SCREEN section of a TRS or in the HEADING, DETAIL, PREVIEW or SUMMARY sections of a REPORT instruction file. Exclamation points that are part of a constant enclosed in single quotes or part of a parameter prompt (enclosed in angle brackets) are also not treated as comment delimiters. In ADMINS command files exclamation points are not considered comment delimiters by default, but this can be enabled by using the COMMENT keyword (see Section 14.1 “Preparing A Command File”).

1.3.2 Continuation

In general, continuation lines are not allowed in instruction files. However, there are certain statements which, due to their possible length, allow continuation. In these cases, the line to be continued ends with a blank followed by a colon (" :") and the continued line is indented, i.e. does not begin in column one. Statements which may be continued are noted in each application section. Statements within paragraphs of an RMS are continued simply by indenting each line after the first line of a paragraph. i.e. the colon at the end of the line is not required.

1.3.3 Indirect References

All instruction files may include indirect references to other files. When the instruction file is used, the indirectly referenced file is substituted for the reference itself. Indirect referencing is a way to include common statements in multiple instruction files. Indirect references are signaled by beginning the line with a double "at" sign (@@) followed by a file name. For example,

```
@@heading.inc
```

would be a way of including a standard page heading in all reports.

Indirect references may be "nested", i.e. indirectly referenced files may contain other indirect references, to ten levels of "depth".

1.3.3.1 Passing Parameters in Indirect References

It is possible to pass parameters to indirectly referenced files. On the "@@filename" line you may add a number of parameter values to be substituted in the indirectly referenced text. The substitution is controlled by a set of parameter names given in the PARAMETERS statement of the indirectly referenced file. Each parameter name in the file is replaced with a parameter value given on the "@@filename" line, before it is read as an instruction by an ADMINS command.

For example, given the following line in an ADMINS instruction file:

```
@@NEWCUST.INC AMOUNT CNAME CADDR
```

if the file NEWCUST.INC includes the line:

```
PARAMETERS X_AMT X_NAME X_ADDR
```

then any subsequent occurrence of X_AMT, X_NAME and X_ADDR in NEWCUST.INC would be replaced with AMOUNT, CNAME and CADDR, respectively.

If an RMS[2] contains the line:

```
@@NEWID.INC LASTINV# INV# INVDAT TODAY
```

and if NEWID.INC contains the following:

```
PARAMETERS LASTID THISID DATE1 DATE2
*
LASTID = LASTID + 1 ; THISID = LASTID
DATE1 = DATE2
```

the compiler (CMP) would receive:

```
LASTINV# = LASTINV# + 1 ; INV# = LASTINV#
INVDAT = TODAY
```

2.See Chapter 9: “CMP: The Record Maintenance Compiler”.

This makes it possible to use NEWID.INC as a "subroutine" or "macro" to calculate the next available number for all kinds of id numbers where you want to increment the last number used by one and put a date into another field. Another .RMS file might include the line:

```
@@NEWID.INC LSTREF REF# RDATE 11-OCT-89
```

in which case the compiler would receive:

```
LSTREF = LSTREF + 1 ; REF# = LSTREF
RDATE = 11-OCT-89
```

To summarize, in the include file, all occurrences of the first parameter name on the PARAMETERS line are substituted by the first parameter following the file name on the "@@filename" line, the second parameter name is replaced with the second parameter, etc. Up to 9 levels of indirect file references are supported (i.e. included files which include other files, to a depth of nine levels).

If more parameter names are found on the PARAMETERS line than there are parameter values supplied on the "@@filename" line, a warning message is displayed and the extra parameters are ignored.

If fewer parameter names are found on the PARAMETERS line than there are parameter values supplied on the "@@filename" line, a warning message is displayed and the extra parameters are ignored. To supply a null value for a parameter name, enter two consecutive apostrophes on the "@@filename" line. For example, the following include file can be used in either the field declaration portion of a TRS or the local section of an RMS:

```
*
* OPFIELDS.INC: Including optional fields in TRS or RMS.
*
* Parameters:
*
* $ER$           the field declaration code, with
*                trailing blank, i.e 'ER ' or 'DR '
*                (used in TRS, null in RMS)
* $PREFIX$       the field name prefix all
*                the optional fields will share.
* $EXCLAMATION$  value of '!' comments out the rest
*                of the line in RMS (null in TRS)
* $LOCATION$     in TRS, precise placement line
*                "anchor" for 4 fields (null in RMS)
*
PARAMETERS $ER$ $PREFIX$ $EXCLAMATION$ $LOCATION$
*
$ER$$PREFIX$_IN/D2 $EXCLAMATION$ [$LOCATION$,10,20]
$ER$$PREFIX$_OUT/D2 $EXCLAMATION$ [$LOCATION$+1,10,20]
$ER$$PREFIX$_BEG/D2 $EXCLAMATION$ [$LOCATION$+2,10,20]
$ER$$PREFIX$_END/D2 $EXCLAMATION$ [$LOCATION$+3,10,20]
```

Including the file in an RMS, as follows:

```
FILE N.MAS
LOCAL
.
.
.
@@OPFIELDS.INC '' 'JAN' '!' ''
```

would cause CMP to process the following lines:

```
FILE N.MAS
LOCAL
.
.
.
JAN_IN/D2 ! [,10,20]
JAN_OUT/D2 ! [+1,10,20]
JAN_BEG/D2 ! [+2,10,20]
JAN_END/D2 ! [+3,10,20]
```

(Note how the exclamation point, a comment delimiter, is used to "comment out" the last part of each line, which is intended for use only in a TRS.)

Referencing OPFIELDS.INC in a TRS:

```
N N.MAS 1 NOMSG
E N
.
.
.
@@OPFIELDS.INC 'DR ' 'JAN' '' '12'
SCREEN
N: N-----
.
.
.
END
```

would cause SCREEN to process the following lines:

```
N N.MAS 1 NOMSG
E N
.
.
.
DR JAN_IN/D2  [12,10,20]
DR JAN_OUT/D2  [12+1,10,20]
DR JAN_BEG/D2  [12+2,10,20]
DR JAN_END/D2  [12+3,10,20]
SCREEN
N: N-----
.
.
.
END
```

Parameter values can be passed to another "nested" indirect file. If an instruction file contains the line:

```
@@INCL1.FIL AMOUNT INVSUM INVDATE
```

and INCL1.FIL contains

```
PARAMETERS X_AMT X_SUM X_DATE
.
.
@@INCL2.FIL X_AMT NAME
```

This would have the same effect as if the @@ line read:

```
@@INCL2.FIL AMOUNT NAME
```

Care should be taken when naming the parameters to make sure that they will not cause unwanted substitutions in the include file. Consider the following example:

```
PARAMETERS NAME
NAME = 'NAME IT:' ; STAT = ASKSCR(Y,X,NAME)
```

If this file was referenced by e.g.:

```
@@INCL.FIL MYFIELD
```

you clearly would like the result to be:

```
MYFIELD = 'NAME IT:' ; STAT = ASKSCR(Y,X,MYFIELD)
```

Instead, since the parameter NAME is replaced with MYFIELD wherever NAME occurs, you get:

```
MYFIELD = 'MYFIELD IT:' ; STAT = ASKSCR(Y,X,MYFIELD)
```

To obtain the desired result, you must name the parameters such that they do not cause ambiguities, as in the following:

```
PARAMETERS X_NAME
X_NAME = 'NAME IT:' ; STAT = ASKSCR(Y,X,X_NAME)
```

The best way to avoid such conflicts is to use a simple naming convention for parameters: for example, you might append a string such as "&P" or "@P", which contains a normally unused character, to the end of parameter names; or you might begin every parameter name with an underscore.

1.3.4 Parameterization

A substitutable parameter is a character string enclosed in angle brackets within an ADMINS instruction file. These bracketed strings are given special treatment by the ADMINS commands that support parameterization, DEFINE, CMP, COM, REPORT, and SCREEN. When any of these commands reads an instruction file and encounters a string enclosed in angle brackets it will prompt, using the bracketed string, for a value to be inserted at that point in the instruction file.

For example, given the following line in an instruction file:

```
SELECT TYPE EQ <Enter Type Selection>
```

The ADMINS command being run will prompt as follows:[3]

```
Enter Type Selection:
```

If no run time string is provided (the user presses RETURN by itself in response, the ADMINS command will terminate. However, if the string is enclosed by double angle brackets, i.e. "<<Enter Selection>>", the ADMINS command will ignore the entire line which contains the bracketed string.

ADMINS REPORT and COM commands provide for "repetitive" parameterization, i.e. substitutable parameters can be made to re-prompt repeatedly until the user replies with a carriage return.

"Logical" parameterization is also provided. Angle-bracketed strings that begin with the characters "L$" or "L_" can be alternatively satisfied with the contents of a logical name.

3.If the logical name ADM_DIALOGBOX is set to the value “P”, command files (AdmCom) and reports (AdmReport) will prompt in a dialog box, rather than in the command prompt window.

Consult the section for each command for specific details on the use of parameterization in that command.[4]

1.3.5 Referencing Data Dictionary Elements

Anywhere the data type of a data field is specified, (e.g. in a DEF, in the local section of an RMS, or in a CREATE statement in REPORT) a reference can be made instead to a Data Dictionary data element.[5] All the attributes of the referenced data element will be applied to the field. To do this, substitute the Data Dictionary data element name, preceded by the '@' character, for the data type.

For example, the following line in a TRS:

```
DR G$PO#/@PO#
```

would pick up the attributes of the data element PO# at compile time and use them for the field G$PO# in the resultant TRO.

This feature could be used to develop applications where the local fields in screens, reports and procedures can be maintained utilizing the Data Dictionary.

For example, if you have an application where the field PO# is defined as X999999, and PO# is defined in the Data Dictionary, you could write your application such that any local field in any .TRS or .RMS that carries a copy of the PO# field makes a reference to the Data Dictionary data element instead of stating a data type.

If you later needed to change the PO# field's data type definition from X999999 to A12, all you will have to do is change the data type in the Data Dictionary for the data element PO# and recompile all your programs.[6]

This feature also enables the use of automatic Lookup Windows, automatic validation against code lists, and automatic User Help with local fields in TRANS.[7]

4. See Section 2.11 “Parameterization” for DEFINE, Section 7.14 “Parameterization” for REPORT, Section 9.7 “Parameterization” for CMP, Section 14.3 “Parameterization” for COM, and Section 5.17 “Parameterization” for SCREEN.
5. See Appendix I: “ADD: The ADMINS Data Dictionary”
6. You might have to enlarge the display width in screens and reports for fields that reference data element PO# to accommodate an A12 field.
7. See Appendix I: “ADD: The ADMINS Data Dictionary” for details.

1.3.6 Conditional Compilation

ADMINS instruction files (e.g. ".COM", ".RMS", ".REP", etc.) and the TRANS and MANUAL environment files may contain "C style" #if, #ifdef etc. syntax for conditional compilation of code.

All ADMINS commands that read an instruction file "pre-process" the file, interpreting lines that begin with one of the "#" keywords described below:[8]

Keyword	Description
#define name [value]	Replace name with value where value may be any string of text. To include whitespace in value enclose it in quotes. The quotes become part of value. If no value is provided name will have the value 0 (false).
#undef name	Undefine name. Removes name from list of defined names. Subsequent occurrences of name in the file will not be replaced with a value, but rather will be read literally.
#ifdef name	The lines following will be compiled if name is defined. (Only tests whether name is defined, does not test value.)
#ifndef name	The lines following will be compiled if name is not defined. (Only tests whether name is defined, does not test value.)
#if name	The lines following will be compiled if name has the value true (1).
#if !name	The lines following will be compiled if name has the value false (0).
#if expression	The lines following will be compiled if expression is TRUE (non-zero). Expression consists of two names or strings separated by a comparison operator. If both strings are numeric then a numeric comparison is performed otherwise a string comparison is performed. The list of valid operators is: **EQ or == LT or < GT or >** **NE or != LE or <= GE or >=**
#else	The lines following will be compiled if the previous #if, #ifdef, or #ifndef evaluated to false.
#endif	Terminates an #if or #ifdef block.
#noifdef	Disables #ifdef processing until "#setifdef" encountered. None of the above commands are interpreted by the preprocessor after the #noifdef statement.
#setifdef	Re-enables #ifdef processing (after a #noifdef).

8. Comments are not allowed on conditional compilation ("#") command lines.

Conditional compilation (#if and #ifdef and #ifndef) statements may be nested up to 32 levels. The example below shows two levels of nesting.

The names _VMS_ _WIN32_ and _ADMINS_ are always defined.

```
#if _VMS_
```

would be true on a VMS system and false on a WINDOWS system, and

```
#if _WIN32_
```

is false on VMS and true on WINDOWS. The name _ADMINS_ has a value corresponding to the version number of the active release of ADMINS, e.g. _ADMINS_ is replaced with 61 if you are using release 6.1 of ADMINS.

If the following line appeared in an ADMINS command file:

```
DISPLAY ADMINS Version _ADMINS_
```

The pre-processor replaces _ADMINS_ with its value so NATCOM would receive the following line to process:

```
DISPLAY ADMINS Version 84
```

Example:

```
#define CAMBRIDGE 1
FILE N.MAS
LOCAL
...
PROGRAM
rms statements
#if CAMBRIDGE
  Cambridge-specific rms statements
#if _VMS_
  Cambridge on VMS-specific rms statements
#endif
#else
  rms statements if CAMBRIDGE is false
#endif
```

1.3.6.1 Defining Names and Values on Command Line

Names can be defined and assigned values at run time using the D (define) command line qualifier. In the following examples the name "PHYSICAL" would be defined (and have the value "0") for the compilation of LOCATE.RMS, and the name CAMBRIDGE would be defined (and have the value "1") for SCREEN's compilation of TAXINQ.TRS.

```
> cmp -d”PHYSICAL” locate

> scr -d”CAMBRIDGE=1” taxinq
```

1.4 Processing Progress of ADMINS Commands

The example above in Section 1.2.1 “Manual Conventions” shows a line of asterisks displayed by AdmMove to show the user the rate of record movement. Before starting to read the input file, MOVE divides the number of records in the input file by 60. As each sixtieth of the input file is read, MOVE displays an "*" starting at the sixtieth column of the line, and working the display of asterisks back down to the first column as the input file is read. In this manner the on-line user is kept informed of the progress of the MOVE processing.

This facility is included in the AdmMove, AdmSort,AdmMaint, and AdmProd commands to allow the on-line user to follow the progress of the processing.[9]

It is possible to suppress the line of asterisks. This may be desired when the user's terminal is operating at low speed. If the user types "NO *" to the first prompt of the command, the command will re-prompt the first prompt and the asterisks will be suppressed during processing.

AdmMove, AdmMaint, and AdmProd can optionally display an activity indicator (or "progress bar") showing that the command is currently still processing.

To display this progress bar use the command line option ***/progress***:

```
progress=x,y|c
```

/progress=c means center the progress bar on the desktop.

/progress=x,y means display the progress bar at position x,y on the desktop, where x and y are in pixels (0,0 is upper left corner)

1.5 Logging Interactive Sessions

Interactive command dialogue in all ADMINS commands can be recorded in a text editable log file. This log file can also function as an ADMINS command file, so dialogue recorded in a log can be edited if desired, and can be run with COM (see Chapter 14: "Command Files").

To log interactive dialogue, assign the name of a log file to the logical name ADM$LOGFILE. If a log file with that name does not exist, ADMINS will create it. As long as ADM$LOGFILE is assigned, ADMINS will append dialogue lines to the log file.

The dialogue which is recorded consists of command lines which call up ADMINS images, and the responses to ADMINS prompts. (In TRANS, only the command line is recorded). The log file also contains a heading to identify it, and a date and time stamp before each command. The ADM$LOGFILE file only logs interactive sessions, and is ignored if ADMINS is running in a command file.

1.6 Providing Responses for Command Dialogue on the Command Line

All ADMINS commands will accept responses provided on the command line as answers to the prompts that occur in their dialogues. Providing responses on the command line is especially useful when inserting ADMINS command dialogue in scripts intended to be run with multiple shells/command line interpreters, but it also

9. REPORT will also display processing progress through the file using the line of asterisks, but as an option rather than by default (see Section 7.2 "REPORT Statement").

a convenient technique for testing or repetitious use of complex ADMINS command dialogues, as the entire list of responses can be recalled, edited and resubmitted using command line editing.

The special option switch, "--" tells ADMINS that the arguments that follow it are to be interpreted, in order, as responses to prompts in the command's dialogue. Responses that contain more than one word must be quoted (in general, to preserve case and to accommodate special characters, it is good practice to put quotes around all responses).

For example, a command file might contain:

```
move
no *
n.mas
n2.mas
CR
y
```

This dialogue could be specified in a single line:

```
move -- "no *" "n.mas" "n2.mas" "CR" "y"
```

The "--" token must immediately precede the first prompt answer on the command line, and must appear after all other normal command line options and arguments. For example:

```
sort -k
 n.mas
 n2.mas i
```

is written as:

```
sort -k -- "n.mas" "n2.mas i"
```

Commands that take their instruction file on the command line, such as MAINT:

```
maint abc
 n
 y
must be written as:
maint abc -- "n" "y"
```

1.7 File Specification

File specifications in ADMINS usually consist of a file name plus a file type. In all the examples of output in this Manual when a file specification is shown, it is shown in the format "filename.type". In actual on-line output displays from ADMINS the complete file specification is shown.

The file specification given to an ADMINS command may not exceed 255 characters. Usually the host environment supplies the unspecified parts of a full specification using the user's "defaults", e.g. the user's default directory.

1.7.1 ADMINS File Types

Examination of the 3 character file type for an ADMINS file name informs the user as to the contents of the file. The following is a list of common ADMINS file types:

File Type	Description
ACF	An ADMINS command file. A text file that contains a sequence of operations to be executed. For example, "PAYROLL.ACF".
DEF	A file definition instruction file. This is the input to the DEFINE command. For example, “RE.DEF”
FLG	A “field log” file as described in Section 2.10 “Field Logs”, created by DEFINE and maintained automatically by TRANS. For example, “RE.FLG”
HLP	A "help" file. Provides immediate explanatory information to the users of certain ADMINS commands while the command is in use.
IDX	An index file as described in Section 4.5 “SORT Example Creating an Index File”. Index files are created by DEFINE, built by SORT, and maintained by TRANS. For example, "STREET.IDX".
LIS	A list file. These files are created by commands which output print lines such as REPORT. The print lines are directed to a list file and subsequently directed to a printing device. For example, "ADMINSA5.LIS".
MAS	A data file. Typically a "master" file. These files are usually made by the DEFINE command from a "DEF" file. For example, "RE.MAS".
MSG	The ADMINS message file. Contains diagnostic messages which are called and displayed by ADMINS commands.
SAV	A "save" file. The ANALYZER saves all the user's analysis steps in this disk file. For example, "ANALYZB3.SAV".
REP	A report instruction file, which is read by REPORT to produce a report. For example, "PAYROLL.REP".
RMO	A record maintenance object file. These files are the result of CMP compiling the RMS file. RMO files are used by MAINT, the record maintenance processor, or with PROD or TRANS. For example, "ADJUST.RMO".
RMS	A record maintenance instruction file. These files are "compiled" by CMP, the record maintenance compiler. For example, "ADJUST.RMS".

File Type	Description
TAB	A "table" file. These files too are created by DEFINE, but are usually used to hold table data or reference data. For example, "OBJECT.TAB".
TAP	An instruction file for the ADMINS external files commands, AdmFAC and AdmFDA, which describes a record layout. For example, "PAYROLL.TAP".
TMP	Temporary files used by ADMINS. Temporary files are created and automatically named[a] by ADMINS SORT (SORTxx.TMP and OUTPxx.TMP), SCREEN (SCRExx.TMP), REPORT (RPxx.TMP), and CMP (COMPxx.TMP) commands.
TRO	A screen object file. These files are made by SCREEN from the TRS file. TRO files are run by TRANS. For example, "PAY.TRO".
TRS	A screen instruction file. For example, "PAY.TRS".

a. See Appendix C.1.1 “Differences in Print File and Temporary File Naming”

1.8 Dynamic Data File Expansion

Most of the ADMINS commands which can add records to a file, will automatically enlarge the file if an impending overflow condition is detected. The commands which support dynamic file expansion are those that add "batches" of records to a file. These include AdmMove, AdmSort, AdmProd, AdmMaint, AdmAded, AdmFac, AdmIE and AdmMrgFil. In AdmProd, both the lookup and output files are dynamically enlarged. In AdmMaint, output files are enlarged. In the Windows environment AdmTrans will also automatically enlarge Level 3[10] files that are about to overflow, and all ADMINS "batch" commands will automatically enlarge Level 3 fileseven if they are open "multi-user". Level 2 (or earlier) files cannot be enlarged automatically when opened by AdmTrans or for multi-user access.[11]

When AdmMove, AdmSort or one of the other commands listed above detects a file overflow condition, the command automatically enlarges the file by 10% of its current size, or by at least 10 1024-byte disk blocks, whichever is larger. An informational message, "<n> ADMINS blocks added to <file>" is displayed on the terminal, and the command continues processing the file. If the enlarging process fails (for example, the disk is full), then the file is closed, with the records that were added intact.

There is some overhead involved each time a Level 2 file is enlarged. In Level 2 files, the index portion of the file (see Appendix E) is copied from its previous position in the file down to the new end of file.

Automatic file enlargement is in effect by default. However, automatic enlargement of Level 2 files can be disabled by including a "9" in the string assigned to the logical name OPTION.

1.9 Localizing ADMINS

Developers of applications for non-English speaking users can "localize" many ADMINS displays, messages and prompts so that users are presented with these items in their own language. ADMINS provides this capability via the logical name ADM$LOCALE[12], to which should be assigned the path name of a text file that contains the localized versions of the messages and prompts.

10.See Appendix E.1.2 "File Level 3"

11. The ENLARG command described in Section 2.8 "AdmENLARG: Enlarging ADMINS Files" may be used explicitly to enlarge a file.

12. The TRANS environment file (TRANS$ENV) can also be used to localize TRANS messages (see Section 6.17.12 ""Localizing" Messages and Prompts"). ADM$LOCALE is read first and should be assigned at the system level. The environment file will override ADM$LOCALE and should be used for customization for a particular individual or application.

Currently the following prompts and messages may be localized:

- Month names
- Prompts and messages in TRANS
- Prompts and messages in MANUAL
- Prompts regarding file access and i/o
- Answers to some prompts

The format of the records in the ADM$LOCALE file is:

```
aaa###=Message
```

where:

aaa	is a three character code identifying the category of the message. Currently, the following categories are recognized: — mth: Month name (mth001 through mth012, i.e.: mth001=january and mth012=december by default) — tra: TRANS messages/prompts, i.e.: tra000= 'Type SCREEN-NAME, FILE-NAME, or C.R." by default — io_: File i/o and access messages/ prompts
###	is a three digit number identifying a specific message/prompt.

Certain[13] single character responses to prompts can also be localized:

```
answer_yes = y
answer_no = n
answer_wait = w
answer_ignore = i
answer_insert = i
answer_exit = e
```

The file 'adm$dist:admins.msg' contains all the items that currently may be localized (except month names and responses), and may be used as a source to create a localized version.

13. These responses are only with prompts related to concurrency and i/o for ADMINS data files, e.g.: answer_insert is used when TRANS prompts "Enter I to insert".

E.g. for Danish users we could create a file called 'dansk.msg' in the adm$dist directory as follows:

```
        ! ADM$LOCALE file for Danish users.
        mth001=januar
        mth002=februar
        mth003=marts
        mth004=april
        mth005=maj
        mth006=juni
        mth007=juli
        mth008=august
        mth009=september
        mth010=oktober
        mth011=november
        mth012=december
        tra000="Tast skærm-navn, fil-navn eller <CR>."
        tra001="Tryk H for hjælp" tra002="Tryk PREV for forrige
   individ"
        tra003="Tryk NEXT for at gemme dette individ"
        tra004="LFEXIT aktiv, tryk NEXT for at akseptere"
        tra005="Tryk NEXT for at gemme dette individ"
        tra006="Kodeord"
        tra017="Fejl - - - tast (%err)"
        tra029="TRANS FUNKTIONS-NØGLER"
        tra030=Funk
        tra031="Beskrivelse"
        tra032="NØGLE"
        man000="ADMINS Procedures Manual"
        man001="Version %s"
        man002="Tast nøgle for at fortsette..."
```

and make the following logical name assignment:

```
admlcr ADM_LOCALE ADM_DIST:dansk.msg
```

and the messages and month names identified in the file will display in Danish.

1.10 Alternative Collating Sequences

ADMINS supports the full 8 bit character set, including **any collating sequence for printable characters,** with the following restrictions.

1. The collating sequence of the characters with an ASCII value of 32 (decimal) or less (space and below) cannot be changed.
2. All the 256 possible characters **must** be assigned unique collating sequence values in the "collating table",[14] i.e. duplicates are **not** allowed.
3. **All** files used within an ADMINS command session must use the same collating table (i.e. files using different collating tables cannot be mixed).

14. The "collating table" is stored in an ADMINS file with a name in the form ADM$COLLDIR:xx.COL. See Section 2.12 "Alternative Collating Sequences" for a details on how to set up and utilize an alternative collating table.

1.11 ADMINS Messages Facility

The ADMINS message facility is designed to display a brief informative message when an error condition occurs. Some messages are specific to a particular ADMINS command (e.g. DEFINE). Other messages are specific to an ADMINS function (e.g. evaluating expressions). Messages are grouped either by particular command (e.g. defnnn), or by particular function (e.g. expnnn). If the command is reading an ADMINS instruction file, such as DEFINE reading a file definition instruction file (DEF), then the line of text in the instruction file which caused the error is also displayed where possible.

1.11.1 Operation of the Message Facility

Most messages in ADMINS are included in text files that are separate from the command programs themselves.[15] When an ADMINS command detects an error condition, it displays the error message code and the message. If the error was found while parsing a line in an ADMINS instruction file, e.g. a report instruction file (REP), then the line number and the content of the line causing the error are displayed. Using an editor, you can then easily locate the line causing the error to correct the problem.

For example, if you had the following report instruction file called "TOTAL.REP."

```
****** TOTAL.REP ******
*
REPORT TOTAL
         FILE DETAIL.MAS
HEADING
CE   TOTAL REPORT
END
CREATE TOTAL AMT1 + AMT2 + AMT3
DETAIL TOTAL
```

Then, attempting to run this report would result in:

```
$ REPORT TOTAL

rep936 Field type must be specified for a created field
Line 8: CREATE TOTAL AMT1 + AMT2 + AMT3
```

Where appropriate, actual values are inserted into the message text to enhance the meaning of the message, as in the following dialogue.

```
$ REPORT TOTAL XYZ

rep920 REPORT "xyz" not found in REP file "total.rep"
```

15. The error message files may, if desired, be kept in a different disk/directory location than the ADMINS commands. Use the logical name ADM$EMSG to designate this alternative location. If ADM$EMSG is not assigned, ADMINS will expect the error message files to be in the location assigned to the logical name ADM$DIST (the same location as the commands themselves).

1.11.2 Expanded Message Facility

Users can specify the "level" of ADMINS messages they wish to see displayed, based on their level of experience. If the user assigns "1" to the logical name ADM$LEVEL, then briefer messages will be displayed as in the examples above. If the user assigns "0" to ADM$LEVEL or if ADM$LEVEL is not assigned, then in addition to the display above, the full explanation, user action, and Manual reference will be displayed. The full text displayed for message rep936 using the example above in Section 1.11.1 "Operation of the Message Facility" would be as follows:

```
$ REPORT TOTAL

rep936 Field type must be specified for a created field
Line 8: CREATE TOTAL AMT1 + AMT2 + AMT3

Explanation: The syntax of a CREATE statement is:

CREATE new-fieldname/type expression

The field type is a required element in the syntax. The valid field
types are integer (I), decimal (Dn), four-word decimal (Fn), date
(DA), alphanumeric (An),and picture (Xpic).

Reference: ADMINS Procedures Manual - 7.13.1

User Action: Correct the CREATE statement by insuring the field has
a field type.
```

1.12 Logging Events and Fatal Errors

ADMINS commands will log information about the occurrence of certain events, and log information about the occurrence fatal errors in short text files that have names that indicate what time the events occurred.

A value assigned to the logical name **ADM_DIR_LOGEVENTS** tells ADMINS to log information when certain events occur, and the value identifies the folder where ADMINS will log the information. The information is logged in files named according to the scheme EventYYYYMMDDHHmmSS.log, reflecting the time of occurrence.

By default, ADM_DIR_LOGEVENTS logs failed printer spooler attempts. To log other kinds of events, specify the event-types to be logged by including the code for the event-type in the value assigned to the logical name **ADM_MASK_LOGEVENT**.

The following event-type codes are supported.

```
A: Log failed printer spooler attempts
B: Log automatic file extensions (level 3 files only)
```

A value assigned to the logical name **ADM$DIR_LOGFATAL** tells ADMINS commands to log information when a fatal exit occurs, and the value identifies the folder where ADMINS will log the information. ADMINS creates time-stamped files that contain the error messages that occurred at fatal exit, along with user name, EXE name, and TRO/screen name, or REP/report name.

1.13 Host and Operating System Differences

ADMINS may be used in a variety of environments. CPU architectures and/or operating systems may differ from installation to installation, but ADMINS functionality, syntax, and use remains essentially the same. There are, however, some instances where ADMINS must perform similar functions in differing ways to accommodate different hardware/operating system environments. These exceptions are explained in Appendix C: “Platform and Operating System Differences”.

A user of ADMINS must learn some of the characteristics of the hardware/operating system environment in order to understand in a general way how to develop and operate applications. These skills can usually be learned quite quickly on a cookbook basis without requiring any deep understanding on the user's part of the inner workings of the CPU or the operating system.

Because ADMINS is also used as a tool by experienced data processing professionals, we have included "hooks" for the integration of more advanced and technical capabilities in the ADMINS application environment. Descriptions of these more technical facilities are generally found in the appendices of this Manual.

Chapter 2:AdmDefine: Creating Files

The AdmDefine command is used to create an ADMINS data file. AdmDefine reads an instruction file, called a file definition, that describes the record layout for a particular file.[1] A file definition always has a file type of ".DEF", e.g. from NAME.DEF, AdmDefine will create a data file called NAME.XXX, where XXX is the file type specified in the NAME.DEF. Each field to be created in the file is named and given a data type as specified in the DEF.

2.1 Outline of a File Definition (DEF)

The file definition is made up of three parts: the file description, the field descriptions, an optional SELECT statement, and optionally secondary INDEX statements. The outline of the file definition is as follows:

```
File Description
[LOGNAM] FILE_TYPE NRECS [FLGSIZ]
Field Descriptions
FIELD_NAME FIELD_TYPE [KEY/SORT] [DER_OP] [SEC_NAME] ["comment"]
...
SELECT Statement
[SELECT expression]
Alternate Index Descriptions
[INDEX #IDX IDXNAME FIELD1 [FIELD2...]
...
```

2.2 AdmDefine Dialogue and Example

We wish to create a file to store the social security number, name, birthday, age and annual salary of a thousand people. Using a text editor, we create the following file definition, called PEOPLE.DEF:

```
*   people.def
*
mas 1000
ss# x999999999 key1  "social security number"
name a30             "name"
birthday da          "birthday"
age i                "age"
ansaly d2            "annual salary"
```

1. Files can also be defined using the ADMINS Data Dictionary, see Appendix I: "ADD: The ADMINS Data Dictionary".

Use the AdmDefine command and this file definition to create an empty data file:

```
$ define people
DEFSZ: 50 NF: 5  KEYLEN:3  RECSZ: 23  NRECS: 1000
# OF BLOCKS   DATA: 51  INDEX: 10  TOTAL: 61
people.mas created
Indexed file. Keys are: SS#
$
```

If the "AdmDefine" command appears by itself on the command line, AdmDefine will prompt for the file definition name and will display only a brief confirmation message:

```
$ define
DEF File Name:people
people.mas created
$
```

2.2.1 AdmDefine Output Messages

The information displayed by AdmDefine when it creates a file includes the following:

Name	Description
DEFSZ	internal size of the file definition in 16 bit words
NF	number of fields defined
KEYLEN	number of 16-bit words in the key fields
RECSZ	number of 16-bit words required for a physical record
NRECS	approximate number of records which file can accommodate without expanding
DATA	number of 1024 byte blocks to be used for data
INDEX	number of 1024 byte blocks to be used for the built-in index
TOTAL	number of 1024 byte blocks to be used for the total file

2.2.2 AdmDefine in Test Mode

If the user wishes to know how much space will be required for a file, but does not wish to actually reserve the space, AdmDefine can be executed in test mode as follows:

```
$ define people test
DEFSZ: 50 NF: 5  KEYLEN:3  RECSZ: 23  NRECS: 1000
# OF BLOCKS   DATA: 51  INDEX: 10  TOTAL: 61
PEOPLE.MAS CREATED
INDEXED FILE. KEYS ARE: SS#
$
```

No file is created. AdmDefine displays the messages so you can tell how much space would be required to accommodate the specified file. AdmDefine reports that the space required for PEOPLE.MAS would be 61 ADMINS blocks (1024 byte-blocks) or 122 disk blocks (512 byte-blocks).

2.2.3 REDEFINE: Redefine & Convert Existing File

The REDEFINE command line qualifier makes it easier to change the DEF of an existing file. Ordinarily, if the file being defined already exists AdmDefine will exit with the following diagnostic message:

```
$ admdefine people
def980 Unable to create output file,
file "PEOPLE.MAS;1" already exists
```

However, if "REDEFINE" is specified in the command line:

```
admdefine people /redefine
```

then if the file already exists AdmDefine renames the existing file to <filename>_SAV.<type>, creates the new file, then runs a perl script called "redef.pl", that performs a MOVE with CONVERT[2] operation to transfer the data from the old file to the new one. If MOVE completes successfully, the command file deletes the original file and itself. If for any reason MOVE exits with an error status, the command file will delete the new file, rename the original file back to its original name, and display the message "Nothing done." If the file has a field log, the FLG file is renamed to <filename>_SAV.FLG and is not deleted. The user must decide what to do with the data it contains as it may not be valid after the file has been redefined.

2.2.3.1 ADDREDEF: Redefine & Convert Existing using DDID

The command line option/ADDREDEF will causeAdmDefine to open the specified file and find the Data Dictionary ID of the file it was originally defined from, and get its DEF information from that dictionary entry before renaming, defining and restoring the records in the file.

Assume dictionary id FI0216 defines the file DATA:Items.mas. At some point this file is copied to WORK:WorkItems.mas. Assume some changes (e.g. new field names) has been made to FI0216. The command:

```
AdmDefine /ADDREDEF WORK:WorkItems.mas
```

will find the file DDID, FI0216, in the WORK:WorkItems.mas header, and use this to access the new DEF information in the dictionary and then redefine the file.

2.2.4 READONLY qualifier: Block Write Access via GENED

The READONLY command line qualifier sets a flag in the file header that blocks any attempt to open a file for write access using GENED mode in TRANS. If the "READ[ONLY]" qualifier is specified when a file is defined, attempts to open the file for writing in GENED mode result in TRANS immediately closing the file and displaying a message that the file is read-only. To view a file defined READONLY in GENED, the user must append "-R" or "-RX" to the file name, assign "Y" to the logical name ADM$READONLY, or include "H" (uppercase) in the string assigned to the logical name OPTION (any of these causes a read-only file open).[3]

2.See Section 3.2.3 “Move with Generalized Field Type Conversion”

3."-R" and "-RX" file access options are described in Section 19.1 “Modes of File Access”. ADM$READONLY is described in Section 6.4 “Entering or Changing Fields”. Option "H" is described in Appendix A: “Options”.

2.2.5 IXONLY qualifier: Create Index-only file

The IXONLY command line qualifier makes it possible to instruct the ADMINS file system to only maintain the index area of a file, and not the data area.[4]

This option can only be used where all the fields are defined as keys. If this is the case, then an index-only file will use only about half the disk space of a regular ADMINS data file, because the key values are stored only once, in the index, rather than in both the index and in each record.

NOTE

Warning: Since no data records are written to the file, there is no way to regenerate the index from the data records if the index should become corrupted, and consequently this option should **only** be used on files which can be totally derived from another file, e.g. a file used as a secondary index into another file, normally populated through the TRANS INDEX clause.

2.2.6 INIT: Initialize File with a Blank Record

Creating a file that is not empty can be useful in a variety of circumstances and applications. The INIT command line qualifier tells AdmDefine to initialize the file it creates with one blank record:

```
$ define people/init
```

2.3 File Description Line

The first line of a DEF instructs AdmDefine what to use as the file type for the data file to be created, and how much space to reserve for the records in the file. As well AdmDefine can be instructed to place the data file on a specific disk device and/or to create a field log file for the data file defined.

The complete file description line layout is as follows:

```
[LOGNAM] FILE_TYPE NRECS [FLGSIZ]
```

The following sections explain the various elements of the file description line, with required elements appearing first.

2.3.1 FILE_TYPE Specification

The FILE_TYPE is the three character file type that is part of the file specification. AdmDefine will use the file name of the DEF and the FILE_TYPE to create a data file name. For example, if NAME.DEF has a file description line as follows:

```
TAB 100
```

then NAME.TAB is to be created to hold up to 100 records. NAME.TAB will be placed on the default device and directory for the user.

4.If the /IXONLY qualifier is used, a level 2, “single-index” file is always created: INDEX statements in the .DEF are silently ignored.

Any three characters may be used as the FILE_TYPE, but it is recommended that standard ADMINS file types be used for clarity of purpose. They are:

- MAS - A master file.
- TAB - A table file.
- IDX - An index file that indexes a master or table file.
- DER - A file derived from a master or table file.
- FLG - A field log file.

2.3.2 NRECS Specification

The NRECS specification is the number of records to be stored in the file. This number determines the disk space that is reserved when the file is defined. A file would usually be defined with sufficient disk space for expansion. However files may be enlarged as described in Section 2.8 "AdmENLARG: Enlarging ADMINS Files".

The reserved space is for NRECS number of records added to the file in an "optimal" fashion, i.e., by appending. If records were added by random insertion additional space may be required. These issues are more fully discussed in Appendix E: "File Concepts".

2.3.3 LOGNAM Specification

If LOGNAM is present on the file description line, LOGNAM is used as the logical name of the disk device and/or the disk directory on which the file is to be placed. If LOGNAM is not present, the file is placed on a device and directory determined at the time the file is defined (see Section 2.3.3.1 "Utilizing DEFs in Other Directories"). For example, if NAME.DEF has the following file description line:

```
DATA MAS 1000
```

then DATA:NAME.MAS is created. The logical name DATA could be assigned to any disk device (or disk device and directory) attached to the system, for example:

```
> admlcr data c:\myfiles\mydata
```

in which case NAME.MAS is created in the folder c:\myfiles\mydata when DATA:NAME.MAS is created.

2.3.3.1 Utilizing DEFs in Other Directories

If the DEF of the file to be created is not in the user's current default directory, the file is created in the same directory as the DEF, as in the following examples:

```
> cd
c:\dev
> admlcr accounts d:\prosys\accounts
> define accounts:people
DEFSZ: 50 NF: 5  KEYLEN:3  RECSZ: 23  NRECS: 1000
# OF BLOCKS   DATA: 51  INDEX: 10  TOTAL: 61
d:\prosys\accounts\people.mas created
Indexed file. Keys are: ss#
>

> define \inv\oct\south
DEFSZ: 162 NF: 35  KEYLEN:30 RECSZ: 36  NRECS: 3000
# OF BLOCKS   DATA: 168  INDEX: 90  TOTAL: 258
\inv\oct\south.mas created
Indexed file. Keys are: region area district repid
>
```

If, however, lowercase "u" is included in the string assigned to the logical name OPTION, AdmDefine will always create the file in the current default folder. Using the first of the two above examples again with "u" in OPTION the file "people.mas" is not created in the folder `d:\prosys\accounts\people.mas` but in the current default folder.

```
> cd
c:\dev
> admlcr option Vu
```

```
> admlcr accounts d:\prosys\accounts
> define accounts:people
DEFSZ: 50 NF: 5  KEYLEN:3  RECSZ: 23  NRECS: 1000
# OF BLOCKS   DATA: 51  INDEX: 10  TOTAL: 61
people.mas created
Indexed file. Keys are: ss#
>
```

Note that any logical name or device name specified inside the DEF (see Section 2.3.3 "LOGNAM Specification") takes precedence over the disk and/or directory specified on the command line, whether or not "u" is in OPTION.

2.3.4 FLGSIZ Specification

If the optional FLGSIZ specification, field log size, is included in the DEF, AdmDefine creates a field log file for the file being defined. The field log is made large enough to hold FLGSIZ records. TRANS, the transaction processor, may then use this field log file to log changes made to fields in the data file. (The field log file layout is described in Section 2.10 "Field Logs". For a detailed discussion of automatic field logging in TRANS, see Section 6.5 "Field Logging") For example, if NAME.DEF had the following file description line:

```
MAS 1000 200
```

then a field log file will be created. The AdmDefine message will include the field log. For example:

```
$ define
DEF FILE NAME:name
NAME.MAS CREATED
NAME.FLG CREATED
$
```

2.4 Field Description Lines

Field description lines are used to specify the name and data type of each field to be included in the records of the file.[5] Field description lines may specify sort control, a derivation operator, and/or one or more secondary names; they may also contain short descriptive comments. The complete field description line layout is as follows:

```
FIELD_NAME FIELD_TYPE [KEY/SORT] [DER_OP] [SEC_NAME] ["comment"]
```

Each of these elements is described in the sections that follow.

2.4.1 Field Names

The first element on each field description line is the name of the field. Each field in the record must have a name. When choosing field names the user should be aware of the following points.

5. The field name may be delimited from the data type by either a blank space or the slash character, "/". In this document the blank space is used.

1. Field names may be up to 18 characters in length. However, names should be kept as small as is consistent with satisfactory documentation, particularly in files with a large number of fields. An average of 6 to 8 characters per name is a good size to aim for, as there is a modest overhead associated with large field names.
2. ADMINS requires that a field name begin or end with an alphabetic character. (This is checked by AdmDefine). That is, #AMT1 is not a valid field name. However, "AMT1" or #AMT" are both acceptable.
3. Do not use parentheses or punctuation characters as part of a field name.
4. Try to make field names within a file unique on the first one (or two or three) character(s). In report and screen layouts, fields to be printed or displayed may be requested by specifying only enough characters to uniquely identify the field. For example:

```
1ADDR A30     "first line of address"
2ADDR A30      "second line of address"
```

   is preferred to:

```
ADDR1 A30      "first line of address"
ADDR2 A30      "second line of address"
```

2.4.1.1 Reserved Field Names

A number of field names have specific meanings or uses in certain ADMINS commands, and consequently should not be used. A list of reserved field names is included in Appendix D: "Reserved Field Names".

2.4.2 Field Data Types

The second element on each field description line specifies the data type of the field. The following list of fields demonstrates the nine possible data types in ADMINS.

```
AGE        I          "age"
ANSALY     D2         "annual salary"
BALANCE    F          "balance"
EMPLDA     DA         "employment date"
NAME       A20        "name"
ACCT#      XA99999    "account number"
MATDATE    DT         "maturity date"
DAYRATE    L2         "distance in meters
TIMESTAMP  TM         "time received"
```

The data types are described below.

- **I - Integer:** The I field type is used for small whole numbers in the range of plus or minus 32,767 (2 to the 15th power minus 1). An integer occupies one 16-bit word or two bytes.
- **Ln - Longword decimal:** The Ln field type is used for larger numbers, outside the range of an integer field type, or when a decimal point is needed. The range for a longword field is plus or minus 2,147,483,647 (2 to the 31st power minus 1). A longword decimal number occupies two 16-bit words or 32 bits or four bytes.

  A longword decimal field may have up to nine decimal places. The number of desired decimal places is specified by a number after the letter L, e.g., "L2". The decimal point is "imaginary", i.e., it is not actually stored in the data.

- **Dn - Decimal:** The Dn field type is used for still larger numbers, outside the range of longword decimal fields. The range for a decimal field is plus or minus 140,737,488,355,327 (2 to the 47th power minus 1). A decimal number occupies three 16-bit words or 48 bits or six bytes.

  Decimal fields also may have up to nine decimal places. The number of desired decimal places is specified by a number after the letter D, e.g., "D3".
- **Fn - Four Word Decimal:** The Fn field type is used for even larger numbers, outside the range of a decimal field type. The range for four word decimal is plus or minus 9,223,372,036,854,775,807 (2 to the 63rd power minus 1). A four word decimal occupies four 16 bit words or 64 bits or eight bytes.

  Four word decimal fields can also have up to nine decimal places. The number of decimal places is placed after the "F" as in "F3".

In choosing among L, D or F fields the following points should be taken into account:

(1) Ignore the decimal point and make a judgment based on the total number of digits. The decimal point is only present on external input and output representations of the data.

(2) REPORT totals D fields into D fields and F fields into F fields. Hence in choosing whether to use a D or F field, take into account the total value of the field for all the records in the file.

- **DA/DT - Date:** The DA and DT field types are used to store dates. Internally, DA fields are coded into one 16-bit word (two bytes), while DT fields are stored in two 16-bit words (four bytes). DA fields can handle dates in the range January 1, 1901 (1JAN1901) to December 31, 2060 (31-DEC-60). DT fields can handle dates for any year in a range from 100 to more than 30000.

  The standard format for display and entry of DT fields is DD-Mmm-YYYY, e.g. 01-Jun-2015. For DA fields, the standard format is DD-MMM-YY, e.g. 01-JUN-15 for dates after the year 2000, and DDMMMYYYY, e.g. 14FEB1981, for dates before. Alternative date formats are specified by assigning a value to the logical name ADM$DATE. (If the logical name ADM$DATE is not assigned, then ADMINS uses the standard format.)

  Only one format can be active at a time.[6]

  Note that the value assigned to ADM$DATE is case sensitive.

  If the value assigned to ADM$DATE contains the upper case characters "M", "D" and "Y" in any order, date fields in ADMINS are to be input and output with two-digit values representing the month and day, and a two-digit (2000 or after) or four-digit (pre-2000) value for the year. Although there are six permutations of these three values, and all six are allowed, only the two that end in "Y" are supported if pre-2000 dates are used. For example:

ADM$DATE	2000 or after example	pre-2000 example
(not assigned)	01-JUN-15	01JUN1951
MDY	060115	06011951
MYD	061501	(not supported)
YMD	150601	"
YDM	150106	"

6. TRANS retranslates ADM$DATE at every field to be displayed if "t" (lowercase) is included in the string assigned to the logical name OPTION (see Appendix A: "Options"). Also, if the TRANS RMO reassigns ADM$DATE (with CRLOG subroutine) at the first RMO call in a screen (BEGREC, UX or AX or IX) the new date format takes effect immediately in that screen.

```
DYM             011506                  "
DMY             010615              01061951
```

In addition to the six formats above, the following date format options are supported:

(1) If the month indicator is lowercase "m" instead of uppercase "M", then the spelling of the month is used in place of the numeric representation. The month is spelled with the first letter in upper case and the remaining letters in lower case (e.g. "January"). If the "m" is followed immediately by a single digit, then only that number of characters of the month is displayed (at least three characters of the month should be displayed). On input, the spelling of the month is case insensitive, and the user only has to input the first three letters of the month. On output ADMINS displays the date with the specified number of characters of the month.

(2) The year indicator "Y" may be followed immediately with a "4" to indicate that the four digit representation is to be used (e.g. 2015) at all times regardless of whether the date is before 2000 or not. Permutations of the year, month and day indicators in which Y4 is not the last argument are allowed, but only if the year argument is separated from the other arguments by literal text (e.g. punctuation) or a space, for example "Y4 M D".

(3) If the day indicator is lowercase "d" then the leading zero will be suppressed for single digit days of the month, e.g. if ADM$DATE is set to "m d, Y4" then "April 6, 2002" is displayed instead of "April 06, 2002".

(4) All characters other than these indicators, including blanks, are treated as literal text on output. On input blanks are ignored, but the literal characters must be present.

The following examples show the output produced when various values are assigned to ADM$DATE.

ADM$DATE	2000 or after example	pre-2000 example
m D, Y4	December 10, 2015	December 10, 1947
M/D/Y	12/10/15	12/10/1947
M-D-Y4	12-10-2015	12-10-1947
Y4 M D	2015 12 10	1947 12 10
Y-M-D	15-12-10	(not applicable)
D m3 Y	10 Dec 15	10 Dec 1947
D-m3-Y4	10-Dec-2015	10-Dec-1947

Remember, when assigning a value to the logical name ADM$DATE, to enclose the value in quotation marks if it includes imbedded blanks or lower case characters. For example:

```
>admlcr adm_date “m d, Y4”[7]
```

If the standard date format is in use, or any alternative ADM$DATE format, where the year is the last element, is in use, dates that are input without a year default to the current year. For example, "1-APR" entered into a date field during the year 2015 would result in "1-APR-15" being stored and displayed.

7. When entering a date, ADMINS first tries to interpret that date using the format specified by ADM$DATE (or, if ADM$DATE is not assigned, the default date format). If ADMINS fails to get a valid date, it checks to see if ADM$DATEIN is defined. If ADM$DATEIN is defined, ADMINS tries to interpret the entered date according to the format specified.
   The output (the displayed date format) always appears as specified by ADM$DATE (or the default date format if ADM$DATE is not assigned).

The logical name **ADM$CENTURY_CUTOFF_YEAR** allows you to control how dates entered with only two digits for the year are interpreted and stored. By default, as described above, ADMINS interprets a two digit value entered for a year to mean a date after the year 2000. That is '1-APR-15' is interpreted and stored as 'April 1, 2015'.

If you assign a two digit value in the range 00 to 99 to the logical name ADM$CENTURY_CUTOFF_YEAR that value is used as the cutoff to determine how to interpret two digits entered for a year in a date field.

For example:

if you make the following logical name assignment

```
>admlcr adm_century_cutoff_year 80"
```

Then any value entered or brought into an ADMINS date field that contains a two-digit year less than 80 is interpreted as being between the years 2000 and 2079. Meanwhile, any value entered or brought into an ADMINS date field that contains a two-digit year of 80 or larger is interpreted as being between the years 1980 and 1999.

The above logical name assignment would allow entry of, for example, a "last inspection date" of "23-JUL-97" (meaning July 23, 1997) and a "next inspection date" of "1-APR-04" (meaning April 1, 2004) in the same screen

- **TM - Time:** The time field type is used to store 24-hour time-of-day values down to ticks, in the format:

  HH:MM:SS.TT

  where HH is hours, with a valid range of 00 to 23, MM and SS are minutes and seconds, each with a valid range from 00 to 59, and TT is ticks, with a valid range of 00 to 99. TM fields are stored in a longword.

  TM fields be entered without punctuation. E.g.

```
110223            is interpreted as 11:02:23.00
1102              is interpreted as 11:02:00.00
11                is interpreted as 11:00:00.00
```

  Two digits must be entered for all subfields, e.g. 123 will be interpreted as 12:03:00.00, not as 1:23:00.00.

- **An - Alphanumeric:** The An field type is used for fixed length strings of text, which may contain numeric, alphabetic and punctuation characters. The type code ("A") is always followed by the number of characters in the field to a maximum of 80, e.g. A20 means a field that can hold 20 alphanumeric characters. Data is always left justified in an alphanumeric field. Each ASCII[8] character occupies one byte or half of a 16 bit word. Since fields are aligned on word boundaries, A5 and A6 both occupy three words.

  (The "^" character in an alphanumeric field is displayed and printed as a blank. This special character is useful in placing leading "blanks" into an input string which ADMINS always tries to left justify on input. "Hats" (^) are also useful in concatenating "blanks" using the concatenation subroutines described in Appendix H.3 "Concatenation Subroutines")

- **Xpic - Pictured:** The Xpic field type is used for codes containing non-numeric characters and numeric digits in fixed positions. The type code X, which stands for "pictured", is always followed by a picture of up to 18 positions

8. The standard coding for alphanumeric characters on most computers.

which shows the layout of the non-numeric characters and numeric digits in the field. The control characters in a picture are "A" for non-numeric and "9" for digit. For example, "XA99999" represents a pictured field containing one alpha followed by five digits as in "C04279". Leading zeroes need not be present on input, e.g. "C4279" is a legitimate input form for "C04279". In general each character position of the picture occupies one byte and each digit occupies one half byte. However characters always start at the next byte. The actual number of words required for a picture varies according to the pattern of "A's" and "9's" in the picture.

There is an option to permit a dash (-) in a digit (9) position. If the logical name OPTION includes the character "F", as described in Appendix A: "Options", a dash (-) may be put in a digit (9) position. This feature was implemented to support "summary levels" in hierarchical coding systems, and its use should be avoided for other purposes.

- **TInn - Internal Text:** TInn fields store documents directly in the text storage file (TSF). Although the size of a text document is limited by physical factors such as available disk storage and available memory, **ADMINS does not limit the size of a document** that can be stored in an internal text field. See Appendix K: "Using Text Fields" for a discussion of the special considerations that are involved when using TInn fields.
- **BLOB - Binary Large OBject:** BLOB fields store objects of any type or format directly in the text storage file (TSF). BLOB fields may only be accessed using the BLOBIO subroutine, described in Appendix H.14.19 "BLOBIO - Access Binary Large Object (BLOB) Field".

  The size of a BLOB is not limited by ADMINS, but by physical factors such as available disk storage and available memory.

2.4.2.1 Input and Output Representation Options

There are several options in ADMINS which alter the representation of the input or output of data fields based on the field type.

1. Negative values in output representation are shown with a leading dash, e.g. "-345.67". If a "P" is included in the logical name OPTION (see Appendix A: "Options") then parentheses are used for negative values in output representation, i.e. "(345.67)". This is applicable to integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types. **This setting is IGNORED** by AdmReport when it is creating CSV or Excel XML format data (see Section 7.24 "Report Command Line Options")

   Alternatively, negative values can be indicated with characters to the right of the number, rather than with a minus sign to the left or parentheses. This is accomplished using the logical name ADM$MINUS (see Section 2.4.2.1 "Input and Output Representation Options"). For example, if the characters "CR" are assigned to ADM$MINUS then "-345.67" is displayed as "345.67CR".
2. The standard input representation of a numeric field includes the comma and decimal point as used in the United States, e.g. "123,456.78". If a "J" is included in the logical name OPTION (see Appendix A: "Options") then the input is accepted with reversal of the comma and decimal point as used in Europe, e.g. "123.456,78". This is applicable to integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types.
3. The standard output representation of a numeric field includes the comma and decimal point as used in the United States, e.g. "123,456.78". If a "K" is included in the logical name OPTION (see Appendix A: "Options") then the output is

presented with the comma and decimal point reversed as used in Europe, e.g. "123.456,78". This is applicable to integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types.

4. Zero values in numeric fields can be suppressed, i.e. displayed as blanks. If a "0" (zero) is included in the logical name OPTION (see Appendix A: "Options") then the output representation of a zero value in a numeric field is a blank. This is applicable to integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types.
5. Commas can be suppressed when numeric fields are displayed in TRANS. If "," (comma) is included in the string assigned to the logical name OPTION (see Appendix A: "Options") then numeric fields are displayed without commas, i.e. 234,541.98 will display as 234541.98. This is applicable to the integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types.

2.4.2.2 Referencing Data Dictionary Data Elements

The field description line may alternatively reference data elements defined in the Data Dictionary[9] instead of explicitly specifying a data type, as follows:

```
FIELDNAME @DD_FIELDNAME
```

The Data Dictionary data element DD_FIELDNAME is referenced by substituting its name, preceded by an '@' character, for the data type specification.

The data type (and all the other attributes, validation and lookup against codelists in TRANS, for example) of data element DD_FIELDNAME will then be picked up from the Data Dictionary and used for FIELDNAME.

2.4.3 Sort and Access Control

The third element on each field description line is the optional key/sort designation. After a field is named and its type is described the user can enter a key/sort designation.

```
FIELD_NAME FIELD_TYPE [KEY/SORT] [DER_OP] [SEC_NAME] "comment"
```

The possible key/sort designations are KEY[n], DKEY[n], ASC[n], DESC[n], (for example, KEY1, DKEY2, ASC3, DESC4, or just KEY, ASC etc.).[10]

KEYn signifies that the particular field is the nth key field of the file. Key fields are used both to order the file and for direct access. For example, if account number is a key field in a budget, specified as follows:

```
ACCT#   X99999999   KEY1
```

then ADMINS can retrieve a record for a particular account number without reading through all the records of the budget file to find the particular account number.

KEYn key fields place the records of the file in ascending order.

Descending key fields[11] are specified using DKEYn, as follows:

```
EMPDATE DA DKEY1
```

9. See Appendix I: "ADD: The ADMINS Data Dictionary"
10. If the key and/or sort keywords are not numbered AdmDefine will assign the proper number.

A file's key may be made up of multiple fields. For example, the budget account number may really consist of three fields: FUND, DEPT and OBJ.

```
FUND  X99   KEY1   "fund number"
DEPT  X999  KEY2   "department number"
OBJ   X999  KEY3   "object of expenditure"
```

In this case the file is in sort order on object within department within fund, and can be directly accessed by these values in that order.

The fields of the key always appear as the first field descriptions in the file definition, that is [D]KEY1 must appear, if it is to appear at all, on the line with the first field name. And then [D]KEY2, if it appears at all, on the line with the second field name. And so on for [D]KEY3, etc.

KEYn and DKEYn fields can be combined in any way:

```
EMPDATE DA DKEY1 "employment date
LNAME A20 KEY2  "last name"
FNAME A10 KEY3 "first name"
```

Files can be sorted[12] beyond the extent of their key fields:

```
LNAME A20 KEY1    "last name"
FNAME A10 ASC2    "first name"
```

The file is to be keyed by last name only, but within last name the file is to be sorted in ascending order of first name.

```
FUND  X99   KEY     "fund number"
DEPT  X999  KEY     "department number"
APPR  D     DESC    "appropriation"
OBJ   X999          "object of expenditure"
```

Here we see a file to be keyed by FUND and DEPT and then sorted within department in descending order of appropriation.[13]

The sort control fields should follow directly after the key fields, and be in order.

Sort control, i.e. ASCn and DESCn, should only be used to prepare files for reports, as in the above example where we list object of expense per FUND/DEPT in high to low order of appropriation. Files that will be accessed directly should only use the [D]KEY[n] techniques.

2.4.3.1 Sequential Files

A file with no keys is called a **sequential** file. A file with keys is called a **keyed** file. A sequential file may be sorted or not, i.e. it may have ASCn or DESCn sort designations.

11. Inequality links, i.e.LINKLT, etc. (see Section 7.13.4.5 “LINK Without an Exact Match”) always work as follows, regardless of the composition of the file's key: "less than" means "toward top of file"; "greater than" means "toward end of file". Thus, in the case of a file with a single descending key, LINKLT will find the record before the record with the key value specified by the link key field, i.e. the record with the next HIGHER key value.
12. Files are "sorted" when the records are in correct ascending or descending order of the specified fields. But because these fields are not KEY fields the records cannot be directly accessed using the files' internal index.
13. Note that when the key or sort keywords have no numbers AdmDefine assigns them in the order the fields are encountered.

Records can **only** be appended to the end of a sequential file. Records can be appended to the end of a keyed file, but records can also be inserted to and/or deleted from anywhere in a keyed file, and also records can be transferred to a new position in the file, i.e. by changing a key value.

Records in sequential files can be accessed only by searching the file sequentially, and therefore they cannot be used with any ADMINS commands or features that access records by key (i.e. by making use of the file's internal index). Sequential files are used in a limited set of special situations. Most applications have no need of them.

Sequential files are used to bypass a corrupted built-in index. This technique is discussed in Section 13.4 "FILECONVERT - Convert ADMINS datafile attributes".

2.4.3.2 Sorting On Significant Bytes

At times we wish that only part of the contents of a field be used to sort the records in the file. This happens when the sort fields are large (e.g. fields with large alphanumeric strings) and we do not wish to exceed the 200 byte key/sort capacity of the sort program; when we know that only part of the field is significant and we wish to speed up the sort process; or when in fact the sort on a partial field is all that the application requires. In these cases we can place a "/n" at the end of the sort designation word. If we do this, then SORT only sorts on the n most significant characters (bytes) of the particular field. Significance is from the left of the field in the case of A and X types, and from the right of the field for D and I types. An example that illustrates most of these principles is a sort of a motor vehicle file by CLASS, MAKE, MODEL, YEAR and IDENTification.

```
*      MVSORT.DEF
* sort vehicles by class, make, model, year, identification
*
DER 20000
CLASS  I    ASC1/1   "vehicle class"
MAKE   A10  ASC2/4   "vehicle manufacturer"
MODEL  A10  ASC3/4   "vehicle model"
YEAR   I    ASC4/1   "year of manufacture"
IDENT  A15  ASC5/5   "vehicle identification"
...
```

As CLASS is a number from 1 to 9 we need only sort on the least significant byte of the five digit integer CLASS field.

MAKE and MODEL names that have the same initial four letters are the same make and model so we need only sort on the first 4 bytes of these two fields.

YEAR is in the range 0-99 (for 1900 to 1999) so the least significant byte[14] is sufficient.

The first five letters of the motor vehicle identification code encodes the manufacturers options, and we wish to list cars of similar options within the CLASS, MAKE, MODEL, YEAR sequencing.

Note that a potential sort string of 39 bytes was reduced to a 15 byte sort string.

14. Any byte contains an integer in the range of 0 to 255, unsigned.

2.4.4 Deriving Aggregates

AdmDefine can also be used to create files that will contain derived aggregations.[15] Deriving aggregates is performed by the SORT command based on the instructions in the DEF. This is described in Section 4.4 "Deriving Aggregates (Summarizing Sort)". Derivation instructions are placed in the file definition to instruct SORT as to which aggregates are to be derived.

The fourth element on the field description line is the optional derivation operator (DER_OP).

```
FIELD_NAME FIELD_TYPE [KEY/SORT] [DER_OP] [SEC_NAME] "comment"
```

The derivation operators, and a short description of their functions, are listed here:

Derivation Operator	(DER_OP)	Function
VALUES	(/V)	Sub-total values
EXISTENCES	(/E)	Tally non-null existences
COUNT	(/C)	Tally all records
AVERAGE	(/AVG)	Average value in run
MAXIMUM	(/MAX)	Find largest value in run
MINIMUM	(/MIN)	Find smallest value in run
FIRST	(/FI)	Take value from first input record in this run
LAST	(/LA)	Take value from last input record in this run
SAME	(/SA)	Take value from same record selected by previous derivation operator (/FI, /LA, /MAX, /MIN)

15. The reporting tool (AdmREPORT) contains comprehensive facilities for aggregating (i.e. taking sub-totals), re-ordering (sorting on any combination of fields), and selecting records and fields, as well as deriving new fields, formatting printout and numerous other features. However, the output from the AdmREPORT is usually a paper printout that is not further computer-usable. By deriving aggregates as files using AdmDefine and SORT, and if necessary computing or recoding new fields using the record maintenance procedures described in Chapter 9: "CMP: The Record Maintenance Compiler" of this manual, the user can produce into files most results that REPORT can produce on printout. These derived files can then be manipulated further by the ADMINS tools.

For example, if we had a telephone directory file (TELFON.MAS) which included the following fields:

```
EXCHNG   X999         "telephone exchange"
NUMBER   X9999        "rest of the telephone number"
```

and we wished to produce a file informing us how many people lived in each telephone exchange, we might prepare the following file definition.

```
*          EXCHNG.DEF
*
* count people per telephone exchange
*
DER 1000
*
EXCHNG  X999 KEY1      "telephone exchange"
#PEOPLE I     -   /E   "number of people in the exchange"
```

The field #PEOPLE will hold the count of the number of "existences" of EXCHNG. The field EXCHNG holds the first 3 digits of the telephone number (the "exchange"). EXCHNG is also a "working field" for #PEOPLE. Working fields are discussed in Section 2.4.4.1 “Method of Operation”.

EXCHNG.DER is created by using SORT to aggregate the records from TELFON.MAS into EXCHNG.DER. A printout of the file EXCHNG.DER after it was aggregated by SORT might be as follows.

```
    EXCHNG.DER

EXCHNG         #PEOPLE

233            7,429
529            8,316
563              714
...
```

The "/FI" and "/LA" derivation operators utilize the order of the records in the input file to determine the value to be placed in the derived (output) record. The "/FI" operator selects the value of the field it is acting on from the first record in the input file with each output file key value. The "/LA" operator selects the last record in the input file for each output file key.

The "/SA" operator allows additional fields to be taken from the "same" record selected by a derivation operator, e.g. "the name of the student in each class with the highest score" could be placed in the output record by including the following in a DEF:

```
SCORE    I       -       /MAX    "highest score"
NAME     A20     -       /SA     "student's name"
```

"/SA" can be used following the "/FI", "/LA", "/MAX", or "/MIN" derivation operators, and always takes values from the same record as selected by the derivation operator that most immediately precedes the "/SA" in the file definition.

Another example. The following aggregation from a detail appropriation file (BUDGET.MAS) would contain the total value, number, average, highest and lowest appropriation per department/fund, as well as the object code of the highest and lowest appropriations in each department/fund.

```
*       BUDSUM.DEF
*    appropriation summaries per department/fund
*
DER  200
FUND      X99   KEY1
DEPT      X999  KEY2
TOTAPPR   D       -     /V     APPR   "total appropriations"
APPR1     D       -            APPR   "working field"
#APPR     I       -     /E             "number of appropriations"
APPR2     D       -            APPR   "working field"
AVGAPPR   D       -     /AVG           "average appropriation"
MAXAPPR   D       -     /MAX   APPR   "maximum appropriation"
MAXOBJ    X99     -     /SA    OBJ    "object code of max. approp."
MINAPPR   D       -     /MIN   APPR   "minimum appropriation"
MINOBJ    X99     -     /SA    OBJ    "object code of min. approp."
```

A printout of the file after it was defined and built by SORT might look as follows.

```
                        BUDSUM.DER

FUND  DEPT     TOTAL   NUMBER    AVG      MAX     MAXOBJ   MIN    MINOBJ
                APPR    APPR     APPR     APPR     CODE    APPR    CODE

01    020     45,000      9     5,000   16,000      08     2,000    53
...
01    530     75,000     10     7,500   44,000      02     4,000    19
...
```

2.4.4.1 Method of Operation

The way these aggregation operations are used to prepare a derived file from an input file is as follows:

1. A derived file definition is created. The key/sort fields in the derived file provide the control for the aggregation operations, i.e. the aggregation is controlled by all of the KEYn, ASCn, and DESCn fields. If an "N" is included in the logical name OPTION (see Appendix A: "Options") during the execution of the SORT, then only the KEYn fields control the aggregation operations.
2. The fields to be operated on from the input file are set up to receive the comparable derived fields in the derived file. Since each field in any file definition must have a name which is unique to that definition, and derived files may aggregate a given input field several ways (e.g. max, min, sub-total), the secondary naming tools are used to set up the field relationships. That is, the input field is set up to go into a derived field containing the appropriate operation code. For example, APPR is moved into MAXAPPR, where APPR is a secondary name for MAXAPPR and "/MAX" is the operation code which instructs SORT to place the largest value of APPR in MAXAPPR for each control break. The operation code is always the fourth element in a field description line.
3. In the case of "/E", "/C", and "/AVG", two fields are required in the derived file definition. The first field is a "working field" and simply receives the input field, i.e. the field which is being counted or averaged. (A key field may be used as a working field, but other aggregation fields may **not** be used as a working field.) The second field contains the operation code (/E or /C or /AVG) and will hold the result, e.g. the number of non-null appropriation values per department, the number of appropriation records per department, the average appropriation per department.

The computation of derived fields is better understood in the context of the operation of the SORT command which performs the derivations as the final step in the sorting process. The operation of SORT is described in Chapter 4: "SORT: Sorting Records Between Files" of this manual. Conceptually, one should imagine file derivation occurring in two distinct stages. In the first stage, the records from the input file are sorted into the order defined by the output file definition, i.e. in output key order. If a SELECT is present in the output file definition it is applied in this stage so that only "selected" records are sorted into the output definition order. Then in the second stage the particular derivation operations are applied to the newly sorted file to produce the derived records for the output file. However the SORT is implemented in such a way (as is made clear in Chapter 4: "SORT: Sorting Records Between Files") that the size of the output file need only be large enough to hold the total number of derived records, not the total number of records in the file. For example, suppose we have a payroll/personnel file of 40,000 records sorted on employee number that includes among its fields the department code and the annual salary. We wish to produce a report showing total salary by department. We could derive a file of departmental aggregates with a DEF as follows:

```
DER  100
#DEPT   X999  KEY1        "department code"
ANSALY  D      -    /V     "annual salary"
STATUS  A1                 "employee status"
SELECT STATUS EQ 'A'
```

The size of the output file need only be large enough to hold a record per department rather than a record per employee. Note the use of the SELECT statement to only include in the aggregation the salary of the active employees (STATUS EQ 'A') in the derived file. The field STATUS, which is not an aggregation field, will have a null value in the final record. The SELECT statement is described in detail in Section 2.5 "Record Selection".

Another common use of derived files is to prepare a table of standard descriptions from a data file. For example, if the payroll/personnel file contained the department name as well as the department record, we could derive a standard table file of departments as follows:

```
TAB  100
#DEPT     X999  KEY1        "department code"
DEPTNAME  A20    -    /FI    "department name"
```

This would give us a list of department names and numbers. We could then edit the names to appear exactly as we would wish them to appear in reports. Then via such facilities as the LINK statement in SCREEN (see Section 5.4.1 "LINK Paragraph") and the TABLE statement in REPORT (see Section 7.13.5 "TABLE Statement") we could use this standard table of department names in screens and reports.

2.4.5 Secondary Field Names

The user has the option of assigning secondary names to fields. The primary name, which is the name that appears first on the line describing the field, is always required. Secondary names are required if certain commands (i.e. MOVE, SORT) are to be used to move data from **comparable** fields that are stored under different names. That is, the data is to be stored under the primary name in the file being defined, but will come from differently named primary fields in other (input) files. A particular field has one primary name and may have several secondary names. For example:

```
LNAME A20 KEY1 LASTNM       "last name"

TELNO A8 - TELEPHONE TNO    "telephone number"
```

In the above examples, LASTNM is a secondary name for LNAME, and TELEPHONE and TNO are secondary names for TELNO. Presumably the "telephone directory" is being created from existing files, which already have data stored by such names as LASTNM, TELEPHONE and TNO. (The dash between the picture and TELEPHONE in the second example is used to occupy the key/sort designation when there is no key/sort field, and other information, e.g. secondary names or derivation operations, is to follow on the line.)

2.4.6 Comments

As has been shown in all the examples, comments may be included on all field description lines by enclosing the comment in quotation marks (") at the end of the line.

In addition, any line with an asterisk (*) in column one is assumed to be a comment line in the file definition and is ignored by the AdmDefine command.

2.5 Record Selection

As should be apparent from our discussion of secondary names AdmDefine is often used to describe files that are to be derived from existing ADMINS files. (Hence the need for secondary names, to relate the new primary field name to other existing, but differently named, fields.)

Record selection criteria can be specified for a derived file using a SELECT statement in the file's definition. The SELECT criteria is applied to the input records of MOVE or SORT operations so that only the specified records are written into the derived (output) file. Only one SELECT statement may be used in a file's definition.

Examples:

```
SELECT TELNO BET 563-0000 AND 563-9999

SELECT OBJ EQ 101 AND APPR GT 10000
```

These two examples might be the record selection criteria in two derived files. One, for those telephone directory entries in exchange "563", and the other for budget item lines where "personal services" (object code "101") have an appropriation in excess of $10,000. The following shows the definition of the "563 exchange" telephone directory, sorted by telephone number.

```
*          EXC563.DEF
* 563 telephone exchange sorted by TELNO
*
DER 5000
TELNO A8 KEY1      "telephone number"
LNAME A20          "last name"
FNAME A10          "first name"
SELECT TELNO BET 563-0000 AND 563-9999
```

The SELECT statement always appears at the bottom of the DEF file and contains an open-ended logical expression involving field names from the DEF and constants. (The rules for forming the logical expressions in ADMINS are described in Section 8 on Expressions.) The SELECT statement can be in excess of one line using the "colon" line continuation convention. That is, to extend the SELECT statement to more than one line place a ":" at the end of the SELECT line which instructs ADMINS to read the next line as part of the SELECT statement.

Example:

```
*          PSGT10K.DEF
* personal services greater than $10K appropriated
*
DER  1000
*
FUND  X99   KEY1     "fund number"
DEPT  X999  KEY2     "department number"
OBJ   X999           "object of expenditure"
APPR  D              "appropriation"
*
SELECT OBJ EQ 101 :
 AND APPR GT 10000
```

2.6 Level 3 File Structure

Over time, the internal structure of ADMINS data files has evolved to accommodate new capabilities and increased capacities. With very few exceptions, when we introduce a new structure all ADMINS tools built after the new structure is introduced continue to recognize the older structures and can use these data files seamlessly.

ADMINS tools built **before** a new structure is introduced, however, cannot be used with the newer data files.

The current data structure for ADMINS is called Level 3. The previous data structure is called Level 2. Structure Level 3 files allow current ADMINS tools to support multiple indices, as described in Section 2.7 “Multiple Indices”. In addition, on Windows, Level 3 files support automatic file enlargement of files opened multi-user, as described in Section 2.8 “AdmENLARG: Enlarging ADMINS Files”. On Windows, AdmDefine creates Level 3 files by default.All ADMINS commands can use (read and write) either Level 2 or Level 3 files[16].

16.See Appendix E: “File Concepts” for details on ADMINS file structures.

2.7 Multiple Indices

An ADMINS Level 3 file can have one primary index (corresponding to the index in pre-level 3 files), and up to 9 alternate (or secondary) indices. To specify alternate indices place[17] an INDEX statement in the .DEF instruction file for each alternate index that is needed. Index statements have the following syntax:

```
INDEX n 'Index name' Field1 [Field 2 …]
```

where n is a number between 1 and 9 (inclusive), and 'Index name' is the description of the index you want users to see when they asks to display the available indices (max. 30 characters long, the apostrophes are only necessary if the name contains white-space).

Instead of an Index Name, a '-' (dash) may be specified, in which case ADMINS automatically "names" the index with a list of its key field names.

Following the index name is a list of fields that makes up the index key (as in the primary index, a maximum of 9 fields totaling a key length of 200 bytes (100 words) is allowed). Add "/D" to a field name to indicate that that field is to be a descending key field, as described in Section 2.4.3 "Sort and Access Control".

In the following example four alternate indexes are specified.

```
MAS 10000
*
CALNUM X999999999 KEY          "Call number"
CALTYP A4                      "Call type"
PRIOR I                        "Priority code"
UNIT A6                        "Unit dispatched for call"
DISPO A4                       "Disposition of call"
1LINE A78                      "Line 1 of comments"
STATUS A1                      "Call status"
CLOSED A1                      "Y = call has been closed"
AREA A4                        "Area"
NUM X99999                     "Street (house) number"
STREET A30                     "Street name"
RECDAT DA                      "Received date"
RECTIM A8                      "Received time"
CLEDAT DA                      "Clear date"
CLETIM A8                      "Clear time"
* Alternate Indexes
INDEX 1 'Call status' CLOSED AREA STATUS PRIOR RECDAT RECTIM
INDEX 2 'Calls by time reported' RECDAT/D RECTIM/D
INDEX 3 'Calls by time cleared'  CLEDAT/D CLETIM/D
INDEX 4 'Calls by street address'  STREET NUM RECDAT/D RECTIM/D
```

NOTE

Note that Multiple Index files can also be specified and created using the ADMINS Data Dictionary, as described in Appendix I.8 "File Alternate Indices screen".

Any alternate index in a multiple-index file can be **active**, **disabled** or **dropped**. An **active** index is an index in good standing, containing valid index entries to all the records in the file. A **dropped** index occurs where the application has positively asked to eliminate the index from the file. The index remains dropped - you have to redefine the file to bring it back. A **disabled** index is an index that is not being maintained - for example, you may want to disable alternate indexes so records can

17. Although the INDEX statement may appear anywhere in the DEF as long as the fields contained in the index have been declared above it, it is a good design practice to keep the alternate indices together at the end of the DEF (before or after a possible SELECT statement).

be appended to a file to save time, or you may want to update the file (including inserts and deletes) without the overhead of maintaining the alternate indices. **AdmSORT** automatically regenerates and reactivates disabled indexes.

2.7.1 Use of Multi-Indexed Files.

Interactive commands like AdmTrans (or AdmAded for that matter) will automatically maintain all active indices as records are being added, deleted or changed.

Maintaining all active indexes automatically is the default behavior for all ADMINS commands, but this might not be desireable for "batch" type commands, such as AdmMaint or AdmMove. When AdmMove adds a significant number of records in to a moderately sized file it would likely be more efficient to disable the indices, add the records, and then regenerate the indices using SORT. But in a large file where only a minor number of records are changed via AdmMaint it may be far more efficient to maintain the indices on the fly.

To provide this needed flexibility AdmMove has a "-SORT" command line option to change the behavior of AdmMove. "-SORT" disables alternate indexes in the output file (if it is opened in exclusive mode) before AdmMove starts processing, and rebuilds all indexesby calling AdmSort when processing is completed.

A new file open option, "-D", may be used to disable any alternate indices when a file is opened in exclusive mode. Unlike the "-SORT" command line switch described above, the " -D" file option leaves the alternate indices disabled when processing is complete. Using "-D" would be more efficient, for example, when doing repeated AdmMove steps to add different batches of records to a file. Rather than maintaining indexes throughout several intermediate steps, AdmMove would simply append records in these steps, and processing would be completed adding an explicit AdmSort step at the end of the procedure to rebuild all indexes.

Multi-indexed files make it possible to access all the records in the file using any index with the same efficiency as using the primary key. Mutiple Index files allow ADMINS commands (AdmReport for example[18]) to present the same data ordered in different ways without having to sort or accessing "external" index files.

To specify that an ADMINS command is to use one of the alternate indices to access a file, put the index number (1-9) in the file specification as a file access option (e.g. N.MAS-M2 means open N.MAS in multi-user mode, and use index 2).

---

18. AdmReport TOTAL statements, other than TOTAL EOF, must be written with special care in order to handle all possible alternate indexes use a single report instruction file. The solution in these cases will usually involve the use of preprocessing instructions ("#ifdef") and/or cleverly invoked "include" files.

    One way to program around the TOTAL on KEY break in REPORT would be to use syntax like:

```
FILE N.MAS-<Index #>
…
@@TOTAL1<Index #>.IND
…
@@TOTALn<Index #>.IND
```

In TRANS a user should normally be able to switch to any index active for the file, unless the necessary fields are not present in the screen, or the developer places restrictions on the use of certain indices. When a user in TRANS (or ADED) asks to see the available indices, the index number and the index named specified in the DEF will be shown. In TRANS, indices defined but not active or possible to use, will be shown grayed.

2.8 AdmENLARG: Enlarging ADMINS Files

The AdmENLARG command is used to increase the capacity of an ADMINS data file to hold additional records. ENLARG is used when automatic file enlargement (see Section 1.8 "Dynamic Data File Expansion") will not be in effect, i.e. when it is not supported[19] or has been disabled (if "9" is in the logical name OPTION automatic enlargement of Level 2 files is disabled).

ENLARG can operate on a single file, or alternatively, ENLARG can be given a list of files to be checked and enlarged if necessary.

The syntax for enlarging a single file is:

```
$ enlarg [-skip] [-wait] file-name | @list_file [number-of-
additional-records]
```

For example, to add 5000 records to TELFON.MAS:

```
$ enlarg telfon.mas 5000
  600 ADMINS blocks added to telfon.mas
```

By default, ENLARG exits with an error message if the file is already open by another user or process:

```
$ enlarg telfon.mas 5000
opn003 File "telfon.mas" already open
%NONAME-F-NOMSG, Message number 00000004
```

To prevent this error condition, which could cause command procedures to terminate abnormally, ENLARG has two optional qualifiers[20]:

Qualifier	Action
-skip	Skip enlarge of file if it is already open
-wait	Wait for file to become available, then enlarge (same as if file-name has "-W" appended to it.)

If the optional "number of additional records" is not supplied ENLARG expands the file by 10%.

The syntax for checking a list of files is:

---

19. Automatic enlargement is not supported on Windows when Level 2 files are being accessed by AdmTrans or when Level 2 files are opened for multi-user writing. On Windows, automatic file enlargement of Level 3 files is always supported (unless disabled).
20. File access options appended to the file-name take precedence over the skip and wait qualifiers. See Section 19.2 "Resolving File Access Conflicts".

```
$ enlarg @list-name
```

Where "list-name" is the name of a text-editable file that contains the list of files to be checked. The format of the records in the file check list is as follows:

```
Filename    Maximum % Full  [ Enlarge by % ]
```

For each file on the list, the user supplies a maximum percentage full beyond which the file should be enlarged (e.g., enlarge STUFF.MAS if it is 80% full or more). Optionally, each file can have an expansion percentage (e.g., expand STUFF.MAS by 20%). The default expansion is 10%. An example of what a ENLARG file check list might look like follows:

```
STUFF.MAS          80    20
ACCTS.MAS          90
INV.MAS            80
DISK3:VEND.MAS     75    25
```

The ENLARG command shows how many ADMINS blocks (1024 bytes per block) are added to the file to accommodate the new records.

There is some overhead associated with using ENLARG, namely a copying of the index part of the data file.

2.9 MKDEF - Create .DEF file from Data File

ADMINS provides a utility , MKDEF.EXE, to allow creation of .DEF instruction files from existing ADMINS data files. MKDEF is especially useful in the circumstance where the original instruction file used to create a data file cannot be located.

MKDEF takes a single argument, the ADMINS data file for which a file definition (.DEF) is to be created. For example:

```
MKDEF n:\mydir\emppro.mas
End of session
```

The ".DEF" file is created in the same folder as the data file.

2.10 Field Logs

TRANS, the ADMINS transaction processor, provides for automatic maintenance of a log (a "field log") of changes made via TRANS to the fields in a file. The field log is an ADMINS file just like any other user created file. AdmDefine creates the field log file when a field log size is placed after the master file size in the DEF as we see in the following example.

```
*              TELFON.DEF
*
*  Telephone directory file definition
*
MAS  20000  5000
*
LNAME    A20   KEY1   "last name"
FNAME    A10   ASC2   "first name"
INITIAL  A1            "middle initial"
TELNO    A8            "telephone number"
```

```
#STREET  A6          "street number"
STREET   A20         "street name"
CITYST   A30         "city and state"
ZIP      X99999      "zip code"
```

This DEF requests a 20,000 record master file, TELFON.MAS, and a 5,000 record field log file, TELFON.FLG. TELFON.FLG will have the following layout:

```
*     Layout for TELFON.FLG
*
CHGDAT DA  ASC1  "change date"
TSEQ   I   ASC2  "transaction sequence #"
DLC    DA        "date of last change"
LNAME  A20       "key field from master file"
FLDNAM A8        "name of changed field"
FLDTYP A2        "data type of changed field (I,D,F,DA,A,X)"
TTYP   A2        "transaction type"
SEQ    I         "sequence for multi-line change"
OLD    A16       "old value"
NEW    A16       "new (changed) value"
```

(If a field log is requested, then the TSEQ, for "transaction sequence number", and DLC, for "date last changed", fields required for relating log entries to master file records, are automatically added to the file definition.)

The "field log" file TELFON.FLG is maintained by TRANS, the transaction processor, when changes are made to the defined file, TELFON.MAS. See Section 6.4 for a detailed discussion of automatic field logging in TRANS.

2.11 Parameterization

Parameterization is a feature that permits user-oriented editing of the file definition (DEF) as it is being read by the AdmDefine command. When the DEF is prepared, strings in the DEF are enclosed in angle brackets to indicate that these names or "prompts" are to be typed by the user when the data file is created by AdmDefine. When the AdmDefine command is run on a DEF containing these parameters, the user is prompted by AdmDefine, as it reads the DEF instruction file, to supply responses before the output data file is created.

If the parameter is enclosed in single angle brackets, e.g. < >, and the user does not supply a run time string, (i.e. the user presses carriage return in response to the prompt), then AdmDefine will terminate with an error message. If however, the parameter is enclosed in double brackets, as in "SELECT <<type selection>>", and the user does not supply a response, then AdmDefine will ignore the entire instruction line which contained the double bracketed string.

For example, given the following DEF

```
*        BUDGETDER.DEF
*  derived budget file
*
DER  1000
*
FUND   X99    KEY1    "fund number"
DEPT   X999   KEY2    "department number"
OBJ    X999           "object of expenditure"
APPR   D              "appropriation"
*
SELECT <ENTER SELECTION>
```

if we wanted to create a file which selected APPR GT 1000, the dialogue which ensues when the AdmDefine command is issued is:

```
$ define budgetder
ENTER SELECTION: appr gt 1000
DEFSZ: 41 NF: 4  KEYLEN:2  RECSZ: 6  NRECS: 1000
    # OF BLOCKS   DATA: 18  INDEX: 8  TOTAL: 26
BUDGETDER.DER;1 CREATED
INDEXED FILE. KEYS ARE: FUND DEPT
SELECTION: SELECT APPR GT 1000
```

2.11.1 Logical Parameters

If the parameter string contained in the angle brackets begins with the characters “L_”, (e.g. <L_fieldname>), then AdmDefine first tries to translate the prompt as a logical name. If the logical name has been assigned in either the process, desktop, or system logical name tables, the user is not prompted for the contents of the parameter. Instead the value of the logical name is substituted for the prompt. Parameters which begin with the characters “L_” and are assigned as logical names are called "logical parameters".

When the logical names exist, the display of logical parameter prompts and their values can be suppressed by assigning the lowercase letter "c" to the logical name OPTION (see Appendix A: “Options”).

If a parameter beginning with “L_” is not assigned as a logical name, then the user is prompted for a value as in standard parameterization (see Section 2.11 “Parameterization”).

Prompting for values when the logical name is not assigned can be avoided entirely by supplying a default value in the parameter string, as follows:

```
<L_MINIMUM=0>
```

Specify the default value for the logical name by appending “=value” to the logical name inside the angle brackets. In the example above if the logical name L_MINIMUM is not assigned, the value “0” will be substituted for the parameter.

2.12 Alternative Collating Sequences

ADMINS supports the full 8 bit character set, including **any collating sequence for printable characters,** with the following restrictions.

1. The collating sequence of the characters with an ASCII value of 32 (decimal) or less (space and below) cannot be changed.
2. All the 256 possible characters **must** be assigned unique collating sequence values in the "collating table" (see below), i.e. duplicates are **not** allowed.
3. **All** files used within an ADMINS command session must use the same collating table (i.e. files using different collating tables cannot be mixed).

The "collating table" is stored in an ADMINS file with a name in the form ADM$COLLDIR:xx.COL. where 'xx' uniquely identifies the collating table being used. **This unique two character identifier for the active collating table must be assigned to the logical name ADM$COLLATE whenever ADMINS commands are used.** The logical name ADM$COLLDIR identifies the disk and directory where the table is found. (If ADM$COLLDIR is not assigned, ADMINS will look for the ADM$COLLATE collating table in the disk and directory specified by the logical name ADM$NAT).

For example, assume "DK" (for Denmark) is assigned to ADM$COLLATE, and "XDSK:[COLLATE]" is assigned to the logical name ADM$COLLDIR. ADMINS will use XDSK:[COLLATE]DK.COL as the collating table.

Whenever DEFINE creates a file the two character identifier of the active collating table is stored in the file header. Whenever ADMINS opens a data file it checks this stored identifier and uses the indicated collating table to process the file.

Make sure that the xx.COL files are always in ADM$COLLDIR (or ADM$NAT), and that they are protected from alteration, corruption, or deletion.

If the indicated collating table cannot be found, ADMINS will exit with a diagnostic message:

```
col003 Can't open collating table XDSK:[COLLATE]DK.COL
```

Similarly, ADMINS will exit with a diagnostic message if the collating table contains errors, or does not match the current setting of the logical name ADM$COLLATE.

```
col002 File with DK collating can't be used: DEFAULT collating
       has been set.
```

Thus, in any ADMINS session, only **one** collating table may be used. (In order to switch to an application using a different collating table, for instance in TRANS, you must exit back to the system prompt, and reassign the value of ADM$COLLATE, before calling up the first screen in the new application.

To convert an existing file from one collating sequence to another, use the FILECONVERT utility (see Section 13.4.3 "Convert Collating Sequence"):

```
$FILECONVERT DK.MAS C
Converting file from DEFAULT to DK collating.
File converted.
TCOL.MAS will be sorted after FILECONVERT is finished
FILECONVERT finished.
Now executing FCVA3.COM...
SORT
Input file....: TCOL.MAS
Output file...: IX
Rebuilding index only. OK? Y
11:44:32.00
11:44:33.45 134 records read 8 blocks 1 section(s)
11:44:33.86 Index rebuilt
```

The file is converted from its current collating sequence to the collating sequence indicated by the logical name ADM$COLLATE.

2.12.1 Manipulating the Alternative Collating Tables

The collating table files are ADMINS data files with the following DEF:

```
ADM$NAT COL 256
POS    I   KEY1      "ASCII position (0-255)"
COL    I             "Collating sequence #"
7BIT   I             "Character to print if 7-bit output"
8BIT   I             "Character to store if 7-bit input"
CHAR   A4            "Characters graphic representation"
```

The POS field **must** be consecutive 0 through 255 (one record for each of the 8-bit ASCII characters).

The COL field must have a unique value between 0 and 255.

The 7BIT field specifies the ASCII decimal value to display if the output device is unable to display 8 bit characters.

The 8BIT field specifies the COLlating value between 0 and 255 to use if the input device is unable to input 8 bit characters.

Sample xx.COL files have been included on the distribution tape for the following collating sequences:

```
DK.COL   Default collating sequence for Denmark
NO.COL   Default collating sequence for Norway
SW.COL   Default collating sequence for Sweden
```

To modify a collating table, use the COLTAB screen (COLTAB.TRO) that comes on the distribution tape. (NULL.COL, which should be copied to ADM$NAT, is a dummy file for COLTAB.TRO.) Copy an appropriate xx.COL into the ADM$COLLDIR (or ADM$NAT) directory, assign the two character identifier you choose to the logical name ADM$COLLATE:

```
$ COPY ADM$DIST:DK.COL ADM$COLLDIR:DK.COL    $ ASSIGN DK
ADM$COLLATE
```

then make your changes, if any, to the collating table in the COLTAB screen.

No records should be deleted from or added to a collating table file.

Chapter 3:AdmMove: Moving Records Between Files

AdmMove is a multi-purpose command used to move records from one data file to one or more other data files. The AdmMrgFil command (see Section 3.6 "Merge Files (AdmMrgFil)") can be used to merge two or more sorted input files with identical record formats into one sorted output file.

3.1 Functions of AdmMove

All of the following functions can be performed simultaneously.

1. Add or remove fields from the records of a file.
2. Change the size of alphanumeric or pictured fields. (All other field types are of standard length.)
3. Combine, with successive calls of AdmMove, two or more files into a single file. The contents of the output file may be different than any or all the files moved into it, or the output file may contain fields with different names but comparable content. Fields from the input record are written to fields with the same (primary or secondary) name in the output record.
4. Build sub-files. That is, select particular records via logical expressions or key values, and place them in a derived file.
5. Move records into several different output files with one call of AdmMove, providing an easy and efficient method to split one file into several files.
6. Perform generalized field type conversions, using the AdmMove/CONVERT function.
7. Execute procedural logic (i.e. an RMO)[1] on the input file. This facility provides generalized control of all output file and record selection criteria, while also providing update capability on the input file.

Operating system utilities can be used if the user wishes simply to make a duplicate copy of a file. These commands, however, do not rebuild the key index and recover deleted record slots. AdmMove performs both these functions when being used to "copy" a file.

1. See Chapter 9: "CMP: The Record Maintenance Compiler"

3.2 AdmMove Dialogue

The following dialogue ensues when AdmMove is called:

```
$ move
Input File....: input-file-spec
Output File...: output-file-spec [I]
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
```

As is shown, AdmMove asks for input and output file specifications. If "I" is added to the output file specification, i.e. "MONTHLY.MAS I" the output file will be initialized (emptied) before the input file records are moved into it.

AdmMove uses the internal file definitions of the specified files, and the responses to its prompts to determine what actions are to be performed.

There are several possible responses to the third prompt,

```
# to move / S[kip] # / K[ey_range] / N[o_list]:
```

1. **Move all records.**

   Press RETURN. All input records will be read and appended to each output file, unless this is prevented by SELECT criteria or by W$W control in an input file RMO. (If the output file has alternate indexes records are inserted instead of being appended to the output file.)

2. **Move n records.**

   Type a number. This number is the maximum number of records that will be appended to the output file. This option to move a specific number of records cannot be used with AdmMove/MULTIPLE.

3. **No list.**

   The user types the letter N (for "no list"). The list of field names present in the input file but not present in the output file(s) **will not be** printed on the terminal (feature described below). After receiving "N" for "no list", AdmMove repeats the prompt:

   ```
   # to move / S[kip] # / K[ey_range] / N[o_list]:
   ```

4. **Skip n records.**

   The user types the letter "S" (for "skip") followed by a blank and then a number. The specified number of selected records are skipped before appending to the output file. This feature combined with (2) allows the user to skip to any sequential position in the input file and then only move a specific number of records from there into the output file. Here, as in (3), the prompt is repeated. This skip option cannot be used with AdmMove/MULTIPLE.

5. **Key range.**

   The user types the letter "K" (for "key range"). The key range specifies the records in the input file which are to be read. Therefore only records within the specified key range will be moved.[2] AdmMove will prompt for the "Start of key range", followed by a prompt for the "End of key range". If the input file has multiple keys, enter all or some of the key values, separated by a blank. Null values are used for minor keys not entered.

   If the user responds to the "Start of key range" prompt with a question mark (?), AdmMove will display a list of the key fields and their field types, and then re-prompt for the starting value.

   AdmMove will accept logical names for the key range prompts. If the response to either prompt begins with the letters "L$" then AdmMove will attempt to translate the response as a logical name. If such a logical name exists, AdmMove will use the string assigned to it as the (low or high) key value, otherwise AdmMove will use the response directly as it usually does.

   When the high key value has been entered, AdmMove repeats the prompt:

   ```
   # to move / S[kip] # / K[ey_range] / N[o_list]:
   ```

3.2.1 AdmMove with Multiple Output Files

If the qualifier "MULTIPLE" appears on the command line AdmMove will prompt for multiple output file specifications, as follows:

```
$ AdmMove/MULTIPLE
Input File....: n.mas
Output File...: n2.mas
Output File...: n3.mas
Output File...: cr
# to move / S[kip] # / K[ey_range] / N[o_list]: cr

FLD is not in N2.MAS
N3.MAS has 200 records
M is not in N3.MAS
OK to continue? y
14:42:33
100 records moved, total 100 records in N2.MAS
47 records moved, total 247 records in N3.MAS
14:43:04.12
```

Any of the output files may be initialized by placing "I" after its files specification, i.e. "N3.MAS I".

A null (carriage return) response at the "Output File...:" prompt tells AdmMove to stop prompting for output files, and the AdmMove dialogue proceeds as described in Section 3.2 "AdmMove Dialogue". If any of the output files already have records, the user is notified of each case and there is a single confirmation prompt.

2. Use of the key range requires that the file be in sort. As explained in Section 3.3 "Operation of AdmMove", AdmMove with key range exits as soon as it encounters a record with a key that exceeds the high key value of the specified range. Any subsequent (out of sort) records with key values in the specified range will not be moved to the output file(s).

Files could be split (or records can be duplicated in multiple files) by utilizing the different SELECT criteria in different output files.

In the above example, output file N2.MAS is initially empty, but N3.MAS starts out with 200 records. N2.MAS and N3.MAS have different SELECT statements, consequently different numbers of records are MOVEd into them.

AdmMove with multiple output files is the functional equivalent of separate MOVEs to each output file, but is much faster to type and to run.

All the facilities of AdmMove are available for use with multiple output files except that:

1. you cannot specify a number of records to move;
2. you cannot use the SKIP records function.

There is a limit of 24 output files (23 if there is an RMO).

3.2.2 AdmMove with RMO

The input file[3] for AdmMove can alternatively be specified as an RMO, as in the following example:

```
$ move/multiple
Input File....: customer.rmo w
Operating on CUSTOMER.MAS
Output File...: active.mas
Output File...: inact.mas
Output File...: cr
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
OK to continue? y
10:10:10
    ********************************************************
111 records moved, total 111 records in ACTIVE.MAS
47 records moved, total 47 records in INACT.MAS
158 records updated in CUSTOMER.MAS
10:10:15
```

There may be situations where two or more output files contain the same field, or the input and output files contain the same field, but you want the RMO to treat the fields differently in the different files. The RMO can separately access fields of the same name in different files by using unique secondary names for each such field. These secondary names are then used for local fields declared in the RMS.

Since the RMO is executed[4] before SELECT(s) in the output file(s) are evaluated, the RMO can selectively control appending to the output file(s) by setting fields which are used in output file SELECT statements.

If you want the RMO to update the input file, specify "W" after the RMO name in the AdmMove dialogue, as in the example above.

---

3. An RMO is not allowed on the AdmMove output file, but the RMO can have access to any field in any of the files. If you want the RMO to set a field in an output file, and the field is not in the input file, just declare the field as a local field in the RMS. See Section Chapter 9: "CMP: The Record Maintenance Compiler" for detailed information on the ADMINS RMO facility.
4. see Section 3.3 "Operation of AdmMove"

AdmMove supports the following special internal fields[5] in the RMO: TODAY, NOW, TICKS, QQ, EXIT, WW, PP, the lookahead fields, NX$fieldname and NX$EOF, and the ADM$RECORDLOCK field for handling locked records.

If W$W is present in the RMO, it must be set at each record where writeback to the input file or output file append(s) are desired.

W$W set to	Action
0	no writeback to input no appends to output(s)
1	perform writeback to input if "W" no appends to output(s)
2	no writeback to input append to output(s) if SELECT(s) are passed
3	perform writeback to input if "W" append to output(s) if SELECT(s) are passed

AdmMove automatically re-sets W$W to zero after processing each record. If W$W is zero or 1 (that is, appends to output files are blocked), the record does not count toward the "number of records to move" or "number to skip", if those options are in use.

3.2.2.1 Test Mode in AdmMove

AdmMove provides a Test Mode for testing RMO steps that operates in exactly the same way that Test Mode operates in MAINT (see Section 10.2.2 "Test Mode Operation").[6]

Test Mode is requested by placing the qualifier "TEST" on the AdmMove command line:

```
$move -test
Input File....:
```

5. These special internal fields (except W$W) function in the same way in both AdmMove and MAINT. TODAY, NOW, and TICKS are described in Section 10.9 "Internal Fields: TODAY, NOW, and TICKS". Lookahead is described in Section 10.11 "Look Ahead: NX$fieldname".ADM$RECORDLOCK is described in Section 10.1.1 "ADM$RECORDLOCK"
6. As in MAINT or PROD, AdmMove Test Mode does not actually append records to output files or update records in input files.

3.2.3 Move with Generalized Field Type Conversion

Generalized conversion[7] between field types in AdmMove can be requested by including the "CONVERT" qualifier on the AdmMove command line:

```
$MOVE/CONVERT
Input File....: n.mas
Output File...: dn.mas
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
N/I will be converted to N/A10 in DN.MAS
M/I will be converted to M/X9999999 in DN.MAS
OK to continue? y
```

As shown in the above example AdmMove/CONVERT displays each field to be converted and prompts "OK to continue" for confirmation before processing the file.

An attempt to convert incompatible field types (i.e. D2 to X999) will cause AdmMove to exit with a diagnostic error message.

3.2.4 SELECT qualifier: Run Time SELECT Criteria

If AdmMove is called with the "SELECT" qualifier on the command line, AdmMove prompts for a record selection expression after opening the input file.

```
$move -select
Input File....: n.mas
Select........: amt gt 345
Output File...: dn.mas
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
OK to continue? y
```

This run time selection criteria will then be applied to the input file record (including RMO local fields, if the input file is an RMO). Unless the "OVERRIDE" qualifier is used (see Section 3.2.5 “OVERRIDE: Ignore Output File Select Criteria”), the run time selection criterion is logically combined with each output file's SELECT statement.[8]

7. Changing the size of an alpha (An) field or conversion between compatible picture field formats do not require the use of the "CONVERT" qualifier. These conversions are always made by AdmMove. Other mismatched field types will cause AdmMove to exit with an error message unless "CONVERT" is specified. Always use CONVERT when moving data between fields with a different number of decimal places.
8. The run time SELECT criteria is logically "AND-ed" with the SELECT statement of the output file, i.e. in order for a record to be moved to the output file it must pass both SELECT criteria.

3.2.5 OVERRIDE: Ignore Output File Select Criteria

If AdmMove is called with the "OVERRIDE" qualifier on the command line, AdmMove will ignore the internal SELECT statement of the output file.

```
$MOVE/OVERRIDE
Input File....: n.mas
Output File...: allrecs.mas
Overriding SELECT in ALLRECS.MAS
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
OK to continue? y
```

The OVERRIDE and SELECT qualifiers may be used in combination, as follows:

```
$move -override -select
Input File....: n.mas
Select........: n gt 500
Output File...: allrecs.mas
Overriding SELECT in ALLRECS.MAS
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
OK to continue? y
```

3.2.6 SORT: Rebuild indexes after records moved

By default, if the output file has alternate indices AdmMove maintains all indices (i.e. it will do inserts instead of appends).

The **-sort** command line switch tells AdmMove to **drop all alternate indices** are before the move starts (thus AdmMove will append records instead of inserting them), and all indices are rebuilt using AdmSort after the move finishes. This will normally be the most efficient way to move large numbers of records into a file and maintain all indices - **provided the output files can be opened in exclusive mode**.

3.3 Operation of AdmMove

If an RMO has been specified for the input file AdmMove first loads the RMO into memory.

After AdmMove receives a response specifying the number of records to move,[9] or a carriage return which means move all records, AdmMove sets up the record movements it will perform.

1. **Setup.**

   AdmMove compares the primary field names from the input file with both the primary and secondary field names from the output file(s). AdmMove copies data from the fields in the input file to fields with the same name in the output file(s). Unless inhibited by the "N" (for "no list") response described above, AdmMove prints out any names in the input file that are not present in the output file(s). If there are such "missing" names, AdmMove will prompt "OK to

9. With AdmMove/MULTIPLE, you must respond with a carriage return (move n records is not allowed)

continue?" before proceeding to the next stage. If the user does not respond with "Y" (for "yes"), AdmMove terminates without performing any record movement.

Next AdmMove inspects the output file(s) to see if they are empty. If any output file already contains records, AdmMove prompts "output-file-spec has nnn records. OK to continue?". Again, if the user does not respond with "Y" (for "yes"), AdmMove terminates without performing any record movement. It should be noted that if the user continues, and the output file has no alternate indexes, the records are APPENDED to the output file(s) and the resulting file(s) may not be in sort order.

NOTE: If the file does have alternate indexes, **AdmMove will maintain all indexes in the output file.** That is, **records are inserted** instead of being appended into the output file.

Finally, AdmMove is ready for the process of record movement. The starting time is printed on the terminal and then the following steps are performed for each input record.

2. **Read input record.**

   The input record is read into a buffer.

3. **Check the key range.**

   If a key range was specified the records are read beginning directly with the starting key value entered. If the record read exceeds the end of key range value entered, AdmMove proceeds to step (10).

4. **Execute the RMO.**

   If an RMO has been specified as the input file it is now executed.

5. **Check the SELECT criteria** (both the run time and the output file criteria).[10]

   If the output file definition contained a SELECT statement, it is now evaluated on the data in the input buffer. (Although the field names in the SELECT logical expression may be from the output definition, and thus are "compiled" by DEFINE without reference to any input file definition, AdmMove "relocates" the field references in the compiled expression so it can be evaluated on input records.) If the SELECT expression evaluates to "false", then AdmMove immediately proceeds to step (2) to read another input record, and does not append anything to the output file for this particular input record.

6. **Check the skip option.**

   If the "skip" option was requested and the specified number of records to skip has been satisfied, AdmMove proceeds to step (7). Otherwise, AdmMove adds to the skip count and immediately proceeds to step (2) to read another input record, and does not append anything to the output file for this particular input record. If the SELECT check (5) above has been postponed because of differences in field type and/or size this "skip" option check is also postponed so that it still occurs after the SELECT check. Note that "skip" cannot be used with AdmMove/MULTIPLE.

7. **Initialize output record.**

   The output record **buffer** is initialized to zero or blank depending on the data types of the individual output fields.

10. If AdmMove/MULTIPLE is in use, or if the SELECT statement involves any field whose type or size is different in the input file and the output file, the SELECT is not evaluated until the data has been moved into the output file buffer.

8. **Move data from input to output record.**

   Data from an input field, if the field name matches a primary or secondary field name and field type for an output field, is moved into that output field. Using secondary names the same input field can be moved into several output fields. Data in a field is moved from left to right. If the output field size is smaller or larger than the input field size (possible **only** with alphanumeric or pictured fields) then the output field is truncated or zero/blank filled. Therefore, using secondary names one can move both complete and partial versions of codes or strings from the input record to the output record. Note that all fields in the output record do not need to receive data from the input record and if not will contain null values (including key fields).

   AdmMove is a logical rather than an arithmetic operation. Remember that the decimal point is not actually present in the internal decimal value. Hence, moving a value from a "D2" type field to a "D4" type field moves the same binary quantity but results in a different value, e.g. "123.45" results in "1.2345". Use AdmMove/CONVERT (see Section 3.2.3 "Move with Generalized Field Type Conversion") to preserve the value in a field of type L, D or F when moving it to a field with a different number of decimal places.

9. **Write output record.**

   The contents of the output record buffer are appended (or inserted if the file has alternate indexes) into the output file.[11] If an end of file indication is not present on the input file, or if only a specific amount of records are to be output and AdmMove is still within that amount, then AdmMove returns to step (2). Otherwise, AdmMove proceeds to step (10).

10. **Close files and exit.**

    Both the input and output files are closed. AdmMove prints the following on-line message for each output file:

    ```
    nnnn records moved, total nnnn records in N2.MAS
    ```

3.4 AdmMove Example

A telephone directory file was created using the following definition:

```
* TELFON.DEF
* Company Telephone Directory
MAS 1000
LNAME   A20   KEY1     "last name"
FNAME   A10   KEY2     "first name"
EXT     X9999          "extension"
TERMDA  DA             "termination date"
```

We wish to expand both the size of the telephone directory file to 5000 entries, and the contents of a telephone directory entry to also include title, home telephone and address. However, we do not wish to include those entries that contain a termination date. (In the SELECT we accept any date "less than" a known low date value.) We prepare the following new file definition.

```
* NEWTEL.DEF
* New Telephone Directory
MAS 5000
```

11. If W$W is present in an input file RMO, its value is checked to see if the record is to be appended or omitted. See Section 3.2.2 "AdmMove with RMO".

```
LNAME      A20   KEY1       "last name"
FNAME      A10   KEY2       "first name"
EXTENSION  X999    -    EXT  "extension - renaming EXT"
TERMDA     DA                "termination date"
TITLE      A30               "title"
HFONE      X9999999          "home phone"
ADDRESS    A20               "home address"
CITYST     A20               "city and state"
ZIP        A5                "zip code"
SELECT TERMDA LT 1-JAN-50
```

We then use DEFINE to create the file NEWTEL.MAS. Then we would use AdmMove as follows:

```
$ move
Input file....: telfon.mas
Output file...: newtel.mas
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
16:10:20.68
     ************************************************************
875 records moved, total 875 records in N2.MAS
16:10:26.52
$
```

The new fields will be blank or zero. We could now use TRANS to enter the additional data into the file.

The line of asterisks displayed by AdmMove shows the on-line user the rate of record movement. It is possible to suppress the line of asterisks. If the user types "NO *" to the "INPUT FILE NAME:" prompt, AdmMove will re-prompt for the input file name and the asterisks will be suppressed during processing.[12]

3.5 VIRTUAL qualifier: Complex Processing Using Instruction File

If the qualifier "VIRTUAL" (may be abbreviated to "/V") appears on the command line AdmMove will accept or prompt for the name of an instruction file:

```
$MOVE/VIRTUAL SOLICIT.MOV

or

$move -v
Instruction file....: solicit.mov
```

The VIRTUAL qualifier tells AdmMove to take its instructions from the specified file rather than the AdmMove interactive dialogue.

AdmMove VIRTUAL is much more than just an alternative to AdmMove's interactive dialogue. AdmMove VIRTUAL has much greater functionality than can be called upon from the dialogue, including the following capabilities, some or all of which may be used in combination.

1. Links, including "link multiple" as in report. Chain linking (and chaining link multiples). Links may be written back.

12. If the character "*" (asterisk) is included in the string assigned to the logical name OPTION, the printing of the line of asterisks to show progress through a file is suppressed in all ADMINS "batch" commands. See Appendix A: “Options”.

2. Insert, delete or append records in one or more external files (similar to append paragraphs in screens).
3. RMO called at points designated in the instruction file. RMO can identify at what point it as been called (using "S$S"). RMO control of progress through the AdmMove VIRTUAL instruction file. (Can be used to "loop" in the AdmMove VIRTUAL instruction file.
4. "Group" portions of the instruction file to more precisely control formation of virtual records.
5. Access to all of the regular AdmMove dialogue functionality (via keywords and instruction file statements).

To suppress stars, the prompt can be answered "NO *", and AdmMove /VIRTUAL will re-prompt for an instruction file name.

```
$MOVE/V
Instruction file....: NO *
Instruction file....: EXPLODE.VMI
```

Enable RMO test mode with AdmMove VIRTUAL by placing the qualifier "TEST" (which can be abbreviated to "T") on the command line:

```
$ move -v -t
Instruction file...:
```

"TEST" is the only other qualifier[13] allowed on the command line when AdmMove VIRTUAL is in use.

3.5.1 Operation of AdmMove VIRTUAL

The "virtual input record" consists of fields in the main file, local fields from the RMO, and the fields linked in via LINK paragraphs.[14]

All other fields receive data from AdmMove's virtual input record. Output fields include: the key fields in LINK paragraphs, the key and W (Write) fields in ADD paragraphs, and the fields of the OUTPUT file.

Ignoring GROUP, BREAK, and CHANGE for a moment, AdmMove VIRTUAL performs the following basic operations:[15]

1. Read a record in the input file.

---

13. Section 3.5.4 "AdmMove VIRTUAL Processing Options" describes how to specify the other AdmMove functions within the AdmMove VIRTUAL instruction file.
14. AdmMove VIRTUAL supports up to 1000 fields in the virtual input record.
15. Some notes on file access: The first time a file is opened, it must be opened using the **most restrictive** access mode which will be required by **any usage.** By default, AdmMove VIRTUAL opens the input file in the same mode as AdmMove's interactive dialogue, in single user mode. But because **AdmMove VIRTUAL actually opens LINKs first** and because **LINKs without WRITE are opened by default in read-only mode;** you may have to override the default access modes if you want to LINK to the input file. For the same reasons, if you link (without WRITE) to the input file, and the input file is to be written back, the links must open the file "-M" or "-RM".

2. Process the statements in the instruction file, performing EXECUTEs, LINKs, and ADDs in the order they appear.[16] **Note:** The ADD file appends, inserts, and deletes take place at this point, **before** the LINKs and the main file are written back.
3. Write back main file and/or LINKs.
4. Append records to OUTPUT files.

If there are no LINK MULTIPLEs, AdmMove reads the next input record and the process continues. If there are LINK MULTIPLEs, AdmMove has work to do before it reads the next input record.

LINK MULTIPLEs are examined, from the bottom of the AdmMove VIRTUAL instruction file up, to see if there are any more records with the same key in any of the LINK MULTIPLE files. If there are more records in a LINK MULTIPLE file with the same key, a new virtual record is created using the linked fields from the next record in that LINK MULTIPLE file, all LINKs, EXECUTEs and ADDs below that LINK MULTIPLE are re-executed and a new record is appended to the OUTPUT file. This functionality produces an **"explode" or Cartesian product** of the files, with the last LINK MULTIPLE varying first.

BREAK can be used in the input file or link file paragraphs to tell AdmMove when to generate control breaks and move records to the OUTPUT file. **There can be only one BREAK (or CHANGE), and it must appear in GROUP 0.**[17] BREAK means, "don't do a control break or output a record until you are about to process the next record in this file."

In the following example BREAK is specified on the input file:

```
FILE N.MAS  BREAK
*
LINK M.MAS MULTIPLE
*
LINK M2.MAS MULTIPLE
...
OUTPUT J.MAS ZERO
```

AdmMove will only break once per input file record, and the output file record produced at this control break could be a "summary" of the M.MAS vs. M2.MAS "explode", i.e. a summary of all the links to M.MAS and M2.MAS generated by the input file record.

To produce output records that summarize the multiple links to M2.MAS based on each link between the main file and M.MAS, use BREAK on the LINK to M.MAS:

```
FILE N.MAS
*
LINK M.MAS MULTIPLE BREAK
*
LINK M2.MAS MULTIPLE
...
OUTPUT J.MAS ZERO
```

CHANGE works exactly like BREAK; but it breaks and outputs a record only when the next record in the main file or LINK MULTIPLE file has a different (full or partial, as specified) key value.

---

16. AdmMove usually processes the instruction file statements in top-to-bottom order. There are exceptions. For example, the RMO can intervene to change the order of processing; and EXECUTE BREAK occurs only at a control break.
17. The GROUP statement is explained in Section 3.5.8.2 “GROUP statement”.

3.5.2 AdmMove VIRTUAL: Instruction File Outline

AdmMove instruction files have the following general outline. The various components are described in detail in the sections that follow.

```
                Input File Statement

FILE <filename or RMO> [WRITE] [BREAK or CHANGE[=keyfield]]
                AdmMove Processing Options

[KEY <start_values> TO <end_values>]
[NRECS <n>]
[SKIP  <n>]
[OVERRIDE]
[CONVERT]
[SELECT <expression> [:]
                Output File Statement

[OUTPUT <filename> [ZERO]
                Link File Paragraph(s)

[LINK[LE|GE|LT|GT] <filename> [WRITE] [REQUIRED] [NULL] [INSERT]
                [MULTIPLE] [BREAK or CHANGE]
[A <fieldname>]
[S <fieldname>]
K <fieldname>
L <fieldname> [<2nd_name>] or <beg_fld> - <end_fld> or *
END]
                Add File Paragraph(s)

[ADD <filename> [INSERT] [UNC_INSERT] [DELETE] [APPEND] [ZERO]
[A <fieldname>]
[S <fieldname>]
K <fieldname>
W <fieldname> [<2nd_name>] or <beg_fld> - <end_fld> or *
END]
               Processing Control Statements

 [GROUP]
 [EXECUTE [S$S]]
```

AdmMove recognizes "*" and "!" to delimit comments in the instruction file. Blank lines are ignored, and tabs or blanks may be used to indent lines in any way you want.

3.5.3 The FILE Statement

The FILE statement names the input file, which may be an RMO (this is the **only** place an RMO can be specified in the AdmMove VIRTUAL instruction file). If the input file is an RMO, use EXECUTE statements[18] to designate the point(s) in the instruction file where the RMO should be called. With an RMO, the **WRITE** keyword in the FILE statement enables writing to the input file, controlled by the special local RMO field W$W.[19]

18. AdmMove/V EXECUTE statements are described in Section 3.5.8.1 "EXECUTE statement: RMO Processing".
19. W$W controls writing to the input and output files, as described in Section 3.2.2 "AdmMove with RMO".

In addition, you may append the keyword **BREAK** or **CHANGE** to the FILE statement. BREAK tells AdmMove that a "control break" is to be generated, and a new record is to be moved to the OUTPUT file(s), **only once for each input file record encountered.** CHANGE tells AdmMove that a control break is to be generated, and a new record is to be moved to the OUTPUT file(s), **only when the key value or the specified partial key value changes** in the input file.[20] Partial key values are specified by identifying the last (lowest) key field to be checked by CHANGE. For example, take a file that has the following three keys:

```
EMPNO    X99999 KEY1   "Employee Number"
APPDATE  DT     KEY2   "Date of Appointment"
APPTIME  TM     KEY3   "Time of Appointment"
```

You could indicate that control breaks should be generated from this file only when the value of APPDAT changes (perhaps to accumulate daily totals for a each employee in an OUTPUT file record) with the following file statement:

```
FILE EMPLOAD.RMO CHANGE=APPDATE
```

3.5.4 AdmMove VIRTUAL Processing Options

All of AdmMove's interactive dialogue file processing features can be specified using AdmMove VIRTUAL instruction file statements,[21] as indicated in the following table:

AdmMove Feature	AdmMove VIRTUAL statement
/SELECT (see Section 3.2.4 "SELECT qualifier: Run Time SELECT Criteria")	SELECT
/OVERRIDE (see Section 3.2.5 "OVERRIDE: Ignore Output File Select Criteria")	OVERRIDE
/CONVERT (see Section 3.2.3 "Move with Generalized Field Type Conversion")	CONVERT
Key_range (see Section 3.2 "AdmMove Dialogue")	KEY
# recs to move (see Section 3.2 "AdmMove Dialogue")	NRECS
# recs to skip (see Section 3.2 "AdmMove Dialogue")	SKIP
/MULTIPLE (see Section 3.2.1 "AdmMove with Multiple Output Files")	OUTPUT (multiple times)

See the outline in Section 3.5.2 "AdmMove VIRTUAL: Instruction File Outline" for the syntax to use with these keywords.

20. Use of the CHANGE keyword requires that the file be in sort.
21. The only AdmMove VIRTUAL command line qualifier permitted (besides "VIRTUAL") is "TEST", which invokes RMO test mode. AdmMove test mode is described in Section 3.2.2.1 "Test Mode in AdmMove".

3.5.5 OUTPUT Statement

AdmMove moves records to the files[22] identified in OUTPUT statements. Data from fields in the virtual input record are moved to fields of the same name[23] in the output record, which is then appended to the output file[24].

The **ZERO** keyword tells AdmMove to initialize (empty) the output file prior to beginning of processing.

3.5.6 Link File Paragraph

Link paragraphs in AdmMove are similar in concept to LINKs in SCREEN (described in Section 5.4.1 "LINK Paragraph") and REPORT (described in Section 7.13.4 "LINK Statement") Fields from the virtual input record (K fields) form key values to identify a particular record in a LINK file. The specified fields from the LINKed file (L fields) then become part of the virtual input record (along with the fields of the input file, "local" fields from the RMO, and any fields from previous LINK statements). Thus fields from previous LINKs can be used in forming the key values for subsequent LINKs ("chain linking").

Each link paragraph begins with a LINK statement, which specifies the type of linking, the file being linked, and options that control several AdmMove VIRTUAL link functions, according to the following syntax:[25]

```
LINK[LE|GE|LT|GT] <filename> [WRITE] [REQUIRED] [NULL]
[INSERT][MULTIPLE] [BREAK or CHANGE] [=prefix]
```

A typical LINK statement might look like this:

```
LINK EMPLOYEE.MAS M R B
```

The LINK keyword in the link statement designates a link that must find an exact match on the specified keys (if only a partial key is specified, the link is made to records that exactly match the partial key).

22. Note that the AdmMove VIRTUAL instruction file equivalent of the AdmMove's interactive dialogue "MULTIPLE" qualifier (see Section 3.2.1 "AdmMove with Multiple Output Files") is simply to name more than one OUTPUT file.
23. As when using AdmMove's interactive dialogue, the primary names of the fields in the input record (with AdmMove VIRTUAL the "virtual input record" consists of the input file, the input RMO local fields, and any LINKed fields) are compared with the primary and secondary names of the output record.
24. As with "interactive" AdmMove, if the OUTPUT statement file has alternate indexes, the record is inserted rather than appended.
25. Because of its length, the syntax outline extends to a second line, but a LINK statement **may not be continued** to a second line. Note that all the keyword options can be abbreviated to their first character.

To specify linking that does not require an exact match, use one of the following keywords in place of LINK:

Keyword in place of LINK	Description
LINKGT - Link Greater Than	Links to the next higher record in the link file even if there is an exact match. If there is none higher, null values are retured for the link fields. This happens when the link key values are equal to or exceed those of the last record in the link file.
LINKGE - Link Greater Than or Equal to	Links to an exact match, or if one is not found, links to the next higher record in the link file. If there is none higher, null values are returned for the link fields. This happens when the link key values exceed those of the last record in the link file.
LINKLT - Link Less Than	Links to the next lower record in the link file even if there is an exact match. If there is none lower, null values are returned for the link fields. This happens when the link key values are lower than or equal to those of the first record of the link file.
LINKLE - Link Less Than or Equal to	Links to an exact match, or if one is not found, links to the next lower record in the link file. If there is none lower, null values are returned for the link fields. This happens when the link key values are lower than those of the first record of the link file.

AdmMove VIRTUAL's link actions are controlled by the following LINK statement keywords:

LINK Statement Keywords	Description
WRITE	Updates existing records in the LINK file. WRITE requires an RMO (to change the value(s) of the L field(s) in the linked record). The EXECUTE call for changing the value to be written back should appear after the LINK paragraph in the MOVE VIRTUAL instruction file (or occur after i.e. EXECUTE BREAK).
REQUIRED	If the link fails, nothing happens.[a] (The virtual record built using this LINK is ignored). If REQUIRED is not present, AdmMove VIRTUAL builds a virtual record with null values for the linked fields of failed links.
NULL	Attempt the LINK even if the key value is null. If NULL is not present, AdmMove VIRTUAL doesn't try LINKs when the key value is null.
MULTIPLE	Process a new virtual record for **every record**[b] in the link file that satisfies the link criteria. AdmMove VIRTUAL may have any number of LINKs with MULTIPLE.
INSERT	If the LINK record does not exist, insert[c] a record with the specified key values into the link file. If the LINK record exists, update it (same action as WRITE). As with WRITE, INSERT requires an RMO. INSERT cannot be used with inexact match links (LINKGT, etc.) or with the MULTIPLE or REQUIRED keywords. NULL can be used with INSERT to force update or insertion of a record with a null key value. INSERT requires that values for **all** the LINK file's key fields be specified **in order** as K fields in the LINK paragraph.
BREAK	Prevents subsequent LINK MULTIPLE paragraphs from generating multiple control breaks, and consequently appending multiple records to OUTPUT files, based in a single record in the current LINK. BREAK modifies the operation of LINK MULTIPLE so that no control break is created **until the next record of this LINK file is processed.** Without BREAK, control breaks are generated and records are appended to OUTPUT files each time any **subsequent** LINK MULTIPLE paragraph links to another record using the same key value, which creates a new virtual input record.

LINK Statement Keywords	Description
CHANGE	Works like BREAK; but delays the control break and production of the OUTPUT file record until a different value of the specified key (full or partial, as determined by the K fields in the LINK paragraph) occurs in the current LINK MULTIPLE file. If key values in the LINK MULTIPLE file are unique, CHANGE is functionally the same as BREAK, but BREAK should be used because it is more efficient.
=prefix	Automatically renames all the fields linked by this paragraph. E.g. if field LNAME is linked in by a paragraph that has =VENDOR_ on its LINK line, then that field is automatically renamed VENDOR_LNAME in the AdmMove/V virtual record.

a. This functionality is similar to that of the LOOKUP file in PROD, as described in Section 11.3 “Lookup File”.
b. A AdmMove VIRTUAL LINK statement that includes both REQUIRED and MULTIPLE (described below) would function in the exact same way as an ordinary PROD operation. I.e. nothing happens if the LINK fails; otherwise each record in the LINK file with the specified key value is combined with the input file record to form a new virtual record.
c. LINKs with INSERT function in the same way as a PROD with WI on the LOOKUP file (see Section 11.6 “Inserting In The Lookup File”).

After the LINK statement an A (Action) field and/or an S (Status) field may be declared. If present, the Action and/or Status fields must be declared first, before the K field(s). One or more K (Key) fields and one or more L (Link) fields **must** be declared. The Link paragraph is terminated by an "END" statement.

```
LINK[LE|GE|LT|GT] <filename> [WRITE] [REQUIRED] [NULL]
[INSERT][MULTIPLE] [BREAK or CHANGE]
[A <fieldname>]
[S <fieldname>]
K <fieldname>
L <fieldname> [<2nd_name>] or <beg_fld> - <end_fld> or *
END
```

A	Provides RMO control of WRITE and INSERT in the LINK paragraph. If an Action field is declard (field type A2) it must be a local field in the RMO. The RMO can set it to “W” to enable writing or inserting the record, or to blank to prevent writing or inserting the record. The Action field is **not automatically blanked out** at every RMO call. (If no Action field is declared, the LINK record is always written.)
S	If a Status field (field type A2) is declared and is a local field in the RMO, AdmMove VIRTUAL sets the Status field to "L" if the link is made successfully, (i.e. if the specified key values are **already present** in the LINK file), and to blank if the link fails.[a]

K	One or more fields must be designated from the virtual input record, in the correct order, to provide the key values used to identify records in the LINK file. For LINKs with INSERT, K fields must be designated to provide values for **all** the keys of the link file. Otherwise, partial keys may be specified. K fields must come before L fields.
L	Designates the fields from the LINKed file that are to be included in the virtual input record. Secondary names may be provided for L fields to rename them locally in the virtual input record. Two optional L field syntaxes are provided for convenience:
L *	Means use all the **non-key** fields in the link file.
L <beg_fld> - <end_fld>	is "through" notation (similar to PROD): "-" means use all fields which lie between the two fields named, in the order of the LINK file definition.

a. If the specified key value is not already in the file, the Status field will be blank even if the LINK paragraph inserts the record.

Secondary names cannot be used with the "*" or "-" syntax's. To rename fields with these syntax's, use the automatic renaming "=prefix" method on the LINK line.

3.5.7 Add File Paragraph

The ADD paragraph is similar in concept to the APPEND paragraph in SCREEN (see Section 5.4.2 "APPEND Paragraph"). ADD paragraphs in the AdmMove VIRTUAL instruction file specify records made up of fields from the virtual input record (the K fields and W fields) that are to be inserted into, deleted from, or appended to the file designated in the ADD statement.

The ADD paragraph begins with an ADD statement which identifies the file to be accessed and designates options that control several AdmMove VIRTUAL ADD functions, according to the following syntax:[26]

```
ADD <filename> [INSERT] [UNC_INSERT] [DELETE] [APPEND] [ZERO]
```

A typical ADD statement might look like this:

```
ADD JRNLENTRY.MAS U Z
```

26. All the keyword options can be abbreviated to their first character. ADD statements **may not be continued** to a second line.

AdmMove VIRTUAL's ADD actions are controlled by the following ADD statement keywords:

ADD Statement Keywords	Description
INSERT	Inserts a record with the designated key value into the ADD file if it's not already present. INSERT requires that values for all the ADD file's key fields be specified **together and in order** as K fields in the ADD paragraph.
UNC_INSERT	Unconditionally inserts a record with the designated key value into the ADD file. UNC_INSERT requires that values for all the ADD files' key fields be specified **together and in order** as K fields in the ADD paragraph.
DELETE	Deletes a record with the designated key value from the ADD file. DELETE requires that value for all the ADD files' key fields be specified **together and in order** as K fields in the ADD paragraph.
APPEND	Append the specified record to the end of the ADD file. K fields need not be specified in ADD paragraphs that can only APPEND records.
ZERO	Initialize (empty) the ADD file prior to the beginning of processing.
=prefix	Automatically provides "secondary names" for all the fields to be written by this paragraph. E.g. if field LNAME is to be written by a paragraph that has =VENDOR_ on its ADD line, then that field is automatically loaded with the value of the field VENDOR_LNAME in the AdmMove/V virtual record.

Any ADD paragraph can do one or more of these operations, controlled by the RMO, as described below.

After the ADD statement the ADD Paragraph may declare an A (Action) field and/or an S (Status) field, and may declare one or more K (Key) fields and one or more W (Write) fields. If present, the Action and/or Status field must be the first fields declared in the Add paragraph. The ADD paragraph is terminated by an "END" statement.

```
ADD <filename> [INSERT] [UNC_INSERT] [DELETE] [APPEND] [ZERO]
[A <fieldname>]
[S <fieldname>]
K <fieldname>
W <fieldname> [<2nd_name>] or <beg_fld> - <end_fld> or *
```

```
END
```

Field Name	Description
A	Provides RMO control of the ADD paragraph action. If the ADD paragraph will always do the same operation (e.g., it will always insert a record) put the keyword for that action (I, U, D, or A) on the ADD line, and the Action field is not needed. If the ADD paragraph may do two or more different operations or will sometimes do no operation, then put all codes for the actions the paragraph can do on the ADD line and declare and Action field (field type A2) in the ADD paragraph and in the local field section of the RMO. The RMO then controls ADD operation by setting the Action field to I, U, D, A or blank. As for LINKs, the Action field **is not automatically blanked out.**
S	If a Status field (field type A2)is declared and is a local field in the RMO, AdmMove VIRTUAL automatically sets the Status field to the Action code of the operation performed by the ADD paragraph. If no operation is performed the Status field is set to blank. The ADD paragraph Status field is useful only for conditional INSERT and DELETE because UNC_INSERT and APPEND operations are always performed.
K	Fields must be designated from the virtual input record, in the correct order, to provide key values for **all the keys** present in the ADD file, **if the ADD file operation is** INSERT, UNC_INSERT, or DELETE. K fields must be declared before any W fields. If the ADD file operation is APPEND then K fields need not be specified at all in the ADD file paragraph.
W	Non-key fields in the ADD file record that are to be written from the AdmMove virtual input record are designated "W", or "write" fields. Secondary names may be provided for W fields to explicitly designate the field in the virtual input record from which the ADD file field is to receive its data. If no secondary name is provided, the W field receives its data from the virtual input record field with the same name. As in the LINK paragraph, two optional W field syntax's are provided for convenience:
W *	Means write all the fields to the ADD file which exist in the virtual input record. When "W *" is used any fields in the ADD file which are not in the virtual input record are given null values.
W <beg_fld> - <end_fld>	Is "through" notation. "-" means fields which lie between the two fields named, in the order of the ADD file definition, are to be loaded from fields of the same name in the virtual input record.

Secondary names cannot be used with the "*" or "-" syntax's. To rename fields with these syntax's:

use the automatic renaming "=prefix" method on the ADD line.

3.5.8 Processing Control

AdmMove VIRTUAL provides two statements for controlling the course of its processing.

EXECUTE statements tell AdmMove at what point(s) in the instruction file to call the RMO. EXECUTE statements can use labels to uniquely identify themselves to the RMO via the "S$S" local RMO field[27]. Thus, RMO logic can be designed specifically for each EXECUTE statement. EXECUTE statement labeling can also be used by the RMO to designate at what point in the AdmMove VIRTUAL instruction file AdmMove should continue processing after the RMO processing completes (see the BRANCH special RMO field description in Section 3.5.8.1 "EXECUTE statement: RMO Processing").

The **GROUP** statement is a delimiter which breaks up the AdmMove VIRTUAL instruction file into groups of statements, allowing more precise control of the formation of virtual records when more than one LINK multiple paragraph is present.

3.5.8.1 EXECUTE statement: RMO Processing

AdmMove VIRTUAL may call an RMO[28] at any number of processing points using EXECUTE statements, which work in a manner similar to EXECUTE statements in REPORT.[29]

Place EXECUTE statements at the desired processing points in the AdmMove VIRTUAL instruction file to call the RMO named in the FILE statement. In the EXECUTE statement you may provide a value to be put into the local RMO field **S$S** each time the RMO is called by that EXECUTE statement.

As with AdmMove's interactive dialogue,[30] RMOs run with AdmMove VIRTUAL support TODAY, NOW, TICKS, QQ, EXIT, W$W, and NX$ fields.

In addition, three special RMO fields are supported only by AdmMove VIRTUAL: **SKIP, ADD,** and **BRANCH.**

AdmMove VIRTUAL Functions	Description
SKIP (field type: I)	If set to a nonzero value in the RMO, all processing steps in the AdmMove VIRTUAL instruction file below the EXECUTE where it is set are skipped. No writeback occurs to the input file or to any LINK files; no records are appended to output files, and no action is taken by ADD paragraphs. SKIP is automatically re-set to zero after every RMO call.

27. The S$S local RMO field may be up to size A18.
28. An RMO is a compiled record maintenance procedure, or Record Maintenance Object. See Chapter 9: "CMP: The Record Maintenance Compiler" for details on RMO syntax and preparation.
29. See Section 7.19 "EXECUTE Statement: RMO Processing"
30. See Section 3.2.2 "AdmMove with RMO"

AdmMove VIRTUAL Functions	Description
ADD (field type: I)	If set to a nonzero value in the RMO, all ADD paragraphs are executed (according to the settings of their action fields, if any) immediately after the RMO call (see remarks below in the GROUP section); and then the RMO is re-called. This process repeats itself until the RMO sets ADD to zero.[a] This facility can be used to output any number of records to any ADD file(s) with a single EXECUTE statement.
BRANCH (field type: An[b])	Provides a generalized facility for dynamically "branching" and "looping" within the AdmMove VIRTUAL instruction file by allowing the RMO to change the normal top-to-bottom sequential order of processing steps. The value placed in BRANCH by the RMO tells AdmMove where in the instruction file to go after executing the current RMO call: to the first statement in the instruction file, or to the first statement in the current processing GROUP; or to some other EXECUTE statement.

a. ADD is not automatically re-set.
b. BRANCH may be up to size A18. If both S$S and BRANCH are used, they must be the same size.

If BRANCH is set to the special value 'BEGREC', AdmMove will go to the first executable statement in the instruction file (i.e. the first EXECUTE, LINK, or ADD statement). If BRANCH is set to the special value 'GROUP', AdmMove will go to the first executable statement in the current GROUP (see explanation of GROUP keyword below). Otherwise, BRANCH can be set to an S$S value specified in one of the EXECUTE statements in GROUP 0 or in the current GROUP, either above or below the EXECUTE where the RMO sets BRANCH, and AdmMove will go to that EXECUTE statement. Wherever in its instruction file AdmMove returns to, AdmMove resumes the normal (top-to-bottom) order of processing starting at that point.

If BRANCH is blank, is not present, or is set to some value other than the above (e.g., a non-existent value of S$S), AdmMove ignores BRANCH, blanks it out, and continues with the next instruction file statement in the normal order of processing.

Whenever BRANCH is set, AdmMove performs any pending writebacks to LINKs before continuing to process the instruction file, because the next statement or paragraph may cause LINKs to other records (if the LINK keys have been changed). For reliability, AdmMove also performs any pending[31] write to the main file when BRANCH is set. These writebacks, if not needed for your application, can be controlled with W$W (main file) and Action fields (LINKs with WRITE).

AdmMove automatically sets BRANCH to blank whenever the RMO has set it.

BRANCH is a powerful facility. Like all "goto" constructs, it should not be used unless needed. Over-use and abuse will result in incomprehensible "spaghetti" programs.

31. A "pending" write is one which is in GROUP 0 or in the current GROUP (LINKs in other GROUPs are ignored here).

BREAK or **CHANGE** can also control when the RMO is called. If the special **EXECUTE BREAK** statement is present in the AdmMove VIRTUAL instruction file, then the RMO is called with 'BREAK' in S$S whenever a break occurs. EXECUTE BREAK calls occur **only** at a break, i.e. at the end of processing for the current virtual record, irrespective of its position in the instruction file. Note that by default, ADD paragraphs do not perform any action at BREAK. The special RMO field ADD can be used in the EXECUTE BREAK call to output records to ADD files at a break (this simulates the action of OUTPUT file(s), but more than one record can be added at any break). Writebacks to the input file and to LINKs are also performed after the BREAK RMO call. These writebacks, if not needed for your application, can be controlled with W$W (main file) and Action fields (LINKs with WRITE).

3.5.8.2 GROUP statement

AdmMove VIRTUAL instruction file may have several LINK MULTIPLE paragraphs. Ordinarily each LINK MULTIPLE record is combined with all the links of any subsequent LINK MULTIPLE, which is in turn combined with all the links of any subsequent LINK MULTIPLE, and so on, to create an "explosion" of the combinations of all the LINK MULTIPLEs. For each combination AdmMove would create a virtual record, and append a record to the output file(s). Take three simple files as follows:

```
N.MAS                M.MAS              M2.MAS
---------             ---------          ---------
Key  Data  Data       Key      Data      Key     Data
INT  LOWC  CAPS       LOWC     UPC       UPC     DEC
3    b     B          a        A         A      5.00
.    .     .          a        B         B      2.00
.    .     .          b        B         B      1.25
.    .     .          b        Y         B      2.23
                      b        Y         Y      9.00
                      y        Y         Y      1.00
                     y       Y          Y     4.00
```

and the following AdmMove VIRTUAL instruction file:

```
FILE N.MAS
*
LINK M.MAS MULTIPLE
K LOWC
L UPC
END
*
LINK M2.MAS MULTIPLE
K UPC
L DEC
END
*
OUTPUT Z.MAS Z
```

The first record in N.MAS links, using field LOWC, to **all the records in M.MAS** that have a key (LOWC) value of "b". Each link to M.MAS loads a new value for UPC into the virtual input record. Three records are linked, the first has a value of "B" for the data field UPC, and the next two have a value of "Y" for UPC. In turn, all the records in M2.MAS that match each M.MAS record's value of UPC are linked to, each link loading a new value for DEC into the input virtual record. This action produces an "explosion" of all the link multiple combinations, as shown in the following table:

INT	LOWC	CAPS	UPC	DEC
3	b	B	B	2.00
3	b	B	B	1.25
3	b	B	B	2.23
3	b	B	Y	9.00
3	b	B	Y	1.00
3	b	B	Y	4.00
3	b	B	Y	9.00
3	b	B	Y	1.00
3	b	B	Y	4.00

Each virtual input record would append a record to Z.MAS.

If you do not want every combination to generate a control break and append a record to the OUTPUT file(s), use the GROUP statements to control the combinations. The GROUP statement is a delimiter which breaks up the AdmMove VIRTUAL instruction file into groups of statements. Anything prior to the first GROUP statement is in group 0; anything between the first two GROUP statements is in group 1, etc. The number of groups is unlimited.

With GROUP, virtual records are formed by processing the group 0 statements, followed by the statements in ONE of the other groups, until there are no more LINKs in the last group. For example:

```
FILE N.MAS
*
GROUP
LINK M.MAS MULTIPLE
K LOWC
L UPC
END
*
GROUP
LINK M2.MAS MULTIPLE
K CAPS
L DEC
END
*
OUTPUT Z.MAS Z
```

First, AdmMove reads a record from the input file, N.MAS, and links to M.MAS (group 1) using the value of field LOWC ignoring, for the time being, group 2 (the link to M2.MAS) and processes the resulting virtual record. AdmMove then links the next record with the same key in M.MAS, processes the resulting virtual record, etc., until there are no more records with that same key in M.MAS.

INT	LOWC	CAPS	UPC	DEC
3	b	B	B	-
3	b	B	Y	-
3	b	B	Y	-

Then, without reading another FILE record, AdmMove starts group 2, and processes virtual records based on all the LINK MULTIPLE links to M2.MAS. (Note that in this example the second LINK links using the field CAPS from the main file N.MAS. All the records in M2.MAS with a key that matches the CAPS value in N.MAS ("B") are now linked to.) During group 2 processing, group 1 LINK fields retain the values from the last link to M.MAS.

INT	LOWC	CAPS	UPC	DEC
3	b	B	Y	2.00
3	b	B	Y	1.25
3	b	B	Y	2.23

When there aren't any more LINK MULTIPLE links to M2.MAS, the next input record is read (from N.MAS) and the process starts over. Writebacks to the input file, as well as LINK W's, EXECUTEs, and ADDs in group 0, are always performed; in addition, all operations in the current group are processed.

In the above example, the use of GROUP causes one link to each record in each of two LINK MULTIPLE files, instead of the an explosion (cross-product) of the two LINK MULTIPLEs.

If LINK MULTIPLE paragraph had also been in group 0 in the example, then the whole process would occur for each same-key-value link to that LINK MULTIPLE file. One or more LINK MULTIPLEs may occur in any group; thus parallel sets of explosions can be specified.

If the special ADD[32] field in the RMO is set, the only ADD paragraphs executed are those in group 0 and those in the current group.

32. See Section 3.5.8.1 "EXECUTE statement: RMO Processing"

3.5.9 AdmMove VIRTUAL Example

The sample application described in the instruction files SOLICIT.MOV and SELECTION.RMS demonstrates several AdmMove VIRTUAL capabilities and features.

The model for this very simplified example is a portion of a fund-raising system. A Donor file contains contact information about contributors. The Hobby file lists the various hobbies contributors have expressed interest in, while the Profession file lists their professions. The Payments file lists pledges and payments for each contributor. We wish to add records to a Solicitations file (to be used to contact previous Donors) based on a specific combinations of hobby with profession, or based on past contributions at or above a specified level. The input "Selection" file is used to flag records that have been marked for delete, and should not be used.

```
* -- solicit.mov ----------------------------------------
FILE SELECTION.RMO BREAK  ! runs on selection file, break
                          ! per input record
EXECUTE PRESEL            ! pre-select vs. selection file
LINK DONOR.MAS            ! donor file information
K ID
L FNAM
L LNAM
L INACTIVE
END

EXECUTE SEL2              ! select using donor fields

EXECUTE BREAK             ! perform inserts to solicit.mas
                          ! when all links for selection
                          ! record are done
ADD SOLICIT.MAS U Z       ! insert 1 rec per selection
A TRIG                    ! per input file record
K ID
W *
END
* -------------------------------------------------------
GROUP                     ! Group 1: first link multiple
       LINK HOBBY.MAS M   ! 1 record per donor per hobby
       K ID
       L HOBBY
       END

       EXECUTE G1         ! set flag based on hobby

 * -------------------------------------------------------
GROUP                     ! Group 2: second link multiple
       LINK PROF.MAS M    ! 1 record per donor per job
       K ID
       L PROF
       END

       EXECUTE G2         ! set flag based on job
 * -------------------------------------------------------
 GROUP                     ! Group 3: 3rd link multiple
       LINK PAYMENT.MAS M ! 1 record per donor per payment
       K ID
       L AMT
       L PAID
       END

       EXECUTE G3         ! accumulate total paid

  * SELECTION.RMS ----------------------------------------
```

```
*
FILE SELECTION.MAS
*
LOCAL
*                    ! Processing control fields
TRIG/A2              ! Trigger field for ADD paragraph
SKIP/I               ! Skip input record
ADD/I                ! Execute ADD paragraph
S$S/A6               ! Status identifies RMO call
*
INACTIVE/A2          ! LINK L fields
HOBBY/A10
PROF/A10
AMT/D
PAID/A2
*
HOBFLG/I             ! Local flags, totals, etc.
PROFLG/I
TOTAMT/D
SELNO/I
*
PROGRAM
* ---------------------------------------------------------
IF S$S EQ 'PRESEL' THEN GOTO PRESEL END  ! Branch using S$S
IF S$S EQ 'SEL2' THEN GOTO SEL2 END
IF S$S EQ 'G1' THEN GOTO G1 END
IF S$S EQ 'G2' THEN GOTO G2 END
IF S$S EQ 'G3' THEN GOTO G3 END
IF S$S EQ 'BREAK' THEN GOTO BREAK END
STOP
* ---------------------------------------------------------
PRESEL: IF DEL NE ' ' THEN SKIP = 1 END ; ! Pre-select:
        STOP                              ! skip flagged recs
* ---------------------------------------------------------
SEL2:   IF INACTIVE NE ' ' THEN SKIP = 1 END ; ! Select using
        STOP                                   ! donor file
* ---------------------------------------------------------
G1:     IF HOBBY EQ 'PHOTO' OR HOBBY EQ 'GARDEN' ! Set hobby
           THEN HOBFLG = 1 END ;                 ! flag
        STOP
* ---------------------------------------------------------
G2:     IF PROF EQ 'DOCTOR' OR PROF EQ 'DENTIST'  ! Set job
           THEN PROFLG = 1 END ;                  ! flag
        STOP
* ---------------------------------------------------------
G3:     IF PAID NE ' ' THEN TOTAMT = TOTAMT + AMT END ;
        STOP                                    ! Total paid
* ---------------------------------------------------------

* Selections:
*   1. Hobby is PHOTO OR GARDEN,
*      AND profession is DOCTOR OR DENTIST.
*   2. Total amount paid is $100 or more
*
BREAK:  IF ADD LT 1 AND HOBFLG EQ 1 AND PROFLG EQ 1 THEN
           SELNO = 1 ;                          ! Select #1
           TRIG = 'U' ;
           ADD = 1
        ELSE
           IF ADD LT 2 AND TOTAMT GE 100 THEN  ! Select #2
              SELNO = 2 ;
              TRIG = 'U' ;
              ADD = 2
           ELSE                          ! Finished with this
              ADD = 0 ;                  ! input record
              TRIG = ' ' ;               ! initialize locals
              HOBFLG = 0 ;
              PROFLG = 0 ;
              TOTAMT = 0
           END
        END ;
        STOP
* ---------------------------------------------------------
```

The RMO first eliminates unwanted input file records (where DEL is not blank) at the EXECUTE PRESEL call using SKIP.

Then, at the EXECUTE SEL2 call, the RMO uses information linked in from the DONOR file (the INACTIVE status) to further narrow the selection of records to be processed.

The BREAK keyword in the FILE statement will cause a special EXECUTE BREAK RMO call every time a new record is read in the input file. The ADD statement will unconditionally insert a record into SOLICIT.MAS whenever the RMO sets the "trigger" field, TRIG.

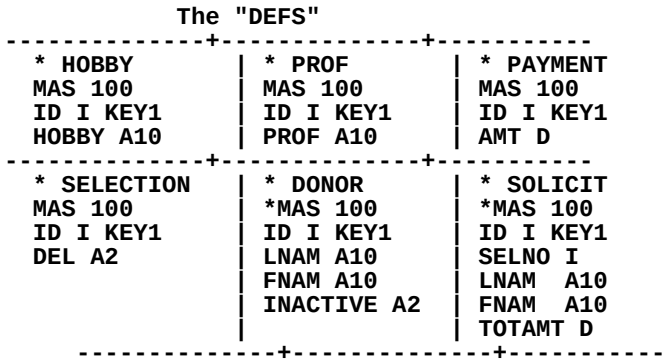
GROUP 1 allows each (selected) input file record to link with **all** the "hobby" records for that donor's ID. If a donor's hobby turns out to be "photo" (photography) or "garden" (gardening) the EXECUTE G1 RMO call will set a flag to indicate that this DONOR has one of the hobbies of interest. GROUP 2 and EXECUTE G2 perform a similar function for "profession" records, searching for Doctors and Dentists.

GROUP 3 and EXECUTE G3 sum up all the paid-up contributions for each Donor.

When all the LINK MULTIPLE links have been performed, AdmMove is finished with the input file record, so it performs the input file BREAK processing. The EXECUTE BREAK RMO call evaluates the data developed by processing the three GROUPs, inserting records into SOLICIT.MAS by setting the "trigger" field whenever either of the following qualifications are met:

1. HOBFLG and PROFLG are both set (the donor has both one of the hobbies and the one of the professions specified).
2. TOTAMT (the sum of all paid-up donations) is at least $100.

The special RMO field ADD is used to re-call the RMO to allow checking (and possible insertion of a record) for each qualification (Note that ID "7" accounts for two of the records that are inserted into SOLICIT.MAS, one because of the hobby/profession combination (photography/dentist); and the second because total contributions are $115.



The Data

Hobbies

ID	HOBBY
1	PHOTO
1	GARDEN
2	SNORKLING
2	TENNIS
3	PHOTO
3	TV
3	GARDEN
4	CHESS
4	SLEEPING
5	COOKING
5	PHOTO
6	EATING
6	TV
7	READING
7	PHOTO
8	CHOIR
8	SPORTS
9	COOKING
9	CHESS
10	PHOTO
10	TV

Professions

ID	PROF
1	TINKER
1	TAILOR
2	SOLDIER
2	SPY
3	DOCTOR
4	CARPENTER
4	DENTIST
5	PEST
5	TEACHER
6	COOPER
6	JUNK
7	CAR SALES
7	DENTIST
8	POLICE
8	TEACHER
9	RECEPTION
9	TYPIST
10	LAWYER
10	PILOT

Payments

ID	AMT	PAID
1	500	X
1	50	X
1	25	X
2	100	X
2	50	
3	10	X
3	200	X
4	25	X
5	100	X
5	25	X
6	75	
6	30	X
7	25	X
7	90	X
8	125	
8	55	
9	45	
9	30	X
10	100	X
10	50	

Donor File

ID	LNAM	FNAM	INACTIVE
0	Zero	Zorro	
1	Horton	Natalie	X
2	Roberts	Tina	
3	Rabbit	Peter	X
4	Feather	Doug	
5	Pacino	Bill	
6	Davis	Avi	
7	Lee	Chuck	
8	Danko	Horace	
9	Spam	Sam	
10	Quill	Jan	

Selection File

ID	DEL
1	
2	
3	
4	
5	X
6	
7	
8	
9	
10	X

Records inserted into SOLICIT.MAS

ID	SELNO	LNAM	FNAM	TOTAMT
2	2	Roberts	Tina	100
7	1	Lee	Chuck	115
7	2	Lee	Chuck	115

3.6 Merge Files (AdmMrgFil)

The MRGFIL command will merge two or more sorted input files into a single output file with the resulting file in sort order. The following is an example of MRGFIL dialogue:

```
$ mrgfil
Merge file....: in1.mas
Merge file....: in2.mas
Merge file....: in3.mas
Merge file....: out.mas
Merge file....: cr
17:35:04.75
**********************************************************
17125 RECORDS WRITTEN INTO OUT.MAS  17:38:16.10
```

Responding to the "Merge file" prompt with just a carriage return indicates that all the files have been entered and that the last file entered is the output file. All other files are input files.

The following restrictions and options apply to the MRGFIL command.

1. All files involved in a merge must have the same defined fields in the same order, i.e., identical file definitions.
2. The output file must be empty at the start of the merge.
3. MRGFIL assumes the input files are "in sort". If the input files are out of sequence, then the resulting output file will be out of sequence.
4. All records in all input files are merged. That is, a SELECT statement in the output file definition is **not** applied during merging of files.
5. The merging of records is done based only on KEY fields and does not consider sort control fields with ASC or DESC designations.
6. Records with the same key value are output in the order that the input files were specified. Using the above example, records with the same key value would be output from IN1.MAS, then IN2.MAS, and finally from IN3.MAS.
7. The line of asterisks may be suppressed by typing "NO *" to the first "Merge file" prompt.[33] MRGFIL will then repeat its prompt.

33. If the character "*" (asterisk) is included in the string assigned to the logical name OPTION, the printing of the line of asterisks to show progress through a file is suppressed in all ADMINS "batch" commands. See Appendix A: “Options”.

Chapter 4:SORT: Sorting Records Between Files

SORT is used to sort or sort and aggregate records from one data file into another.

4.1 Functions of SORT

Like MOVE, SORT moves data from fields in the input file to fields of the same name (or secondary field name) in the output file. As with MOVE, non-key alphanumeric and pic fields in the output file may differ in size from the corresponding field in the input file; and SORT uses the SELECT statement in its output file's definition to choose which input records to process. But SORT differs from MOVE in one fundamental way: the output file is **sorted** using the key and sort designators from the output file definition.

ADMINS is designed primarily to access records in files sorted on their keys (see Section 2.4.3 “Sort and Access Control”). Therefore SORT is usually the appropriate way to "move" the data when the input and output files don't have the same key structure.[1] SORT insures that the file is in sort order on the fields designated as keys in the output file definition, regardless of the order of the records in the input file.

SORT can be used to derive "aggregate" files using the aggregation operators specified in the output file definition.

1. Move appends the records read from the input file into the output file. Thus the sequence of the records in the output file of a MOVE is same as the sequence of the records in the input file. If the output file's key structure differs from the input file, the records cannot be directly found by key value. See Section E.2.1 “Finding Records by Key Value”.

4.2 SORT Dialogue

When SORT is called, it prompts for the names of the input and output files:

```
$ sort
Input file....: input-file-spec
Output file...: [output-file-spec] [I]
```

SORT uses the internal file definitions of the specified files, and the responses to its prompts to determine what actions are to be performed. If the input file is empty, SORT will call a diagnostic message and terminate.

If no output file specification is given, SORT assumes the input file is to be self-sorted and requests a confirmation as follows.

```
Current input file will be deleted after sort. OK?
```

If the user responds with a "Y" for yes, SORT continues in self-sort mode. In self-sort mode, SORT first checks that it can open the file for exclusive use,[2] then SORT creates a temporary output file and sorts the input file into the temporary output file. Finally, the input file is deleted, and the temporary output file is renamed with the name of the input file.

If the user responds to the "Output file" prompt with "IX" or "IX nn" (nn being an integer between 50 and 100) then SORT will perform an "Index-only" self-sort of the file in place, i.e. no temporary file is created and the input file is not deleted. SORT prompts for confirmation as follows:

```
Rebuilding index only. OK?
```

See Section 4.2.2 “Index-only Self-Sort Option” for a detailed discussion of the "Index-only" self-sort option.

If the user types "NO *" to the "Input file" prompt, the asterisks denoting progression through the file will be suppressed and SORT will re-prompt.[3]

The output file of a sort **must be empty.** The SORT output file can be "emptied" by using the Initialize option of the SORT. If an "I" (for initialize) is placed after the output file specification, i.e. "N.MAS I", SORT will initialize (empty) the output file before beginning the sort.

SORT checks to see that the output file is empty. If the output file contains records, and you have not requested that it be initialized, SORT will exit with a diagnostic message.

2. The requirement that the file being self-sorted can be opened for exclusive use cannot be bypassed, i.e. SORT will exit with a diagnostic message if you ask for the file to be opened otherwise or if the file is already open by another user. See Section Chapter 19: “Concurrency Control: Multi-User Files” for a complete discussion of ADMINS file access options.
3. If "*" (asterisk) is included in the string assigned to the logical name OPTION, the printing of the line of asterisks to show progress through a file is suppressed in all ADMINS "batch" commands. See Appendix A: “Options”.

4.2.1 Temporary Files

SORT creates its temporary working file, SORTxx.TMP,[4] in the user's current default directory. In an ordinary self-sort[5], SORT creates its output file, OUTPxx.TMP, in the same directory as the input file. When the sort is finished, the original (input) file is deleted and OUTPxx.TMP is renamed to the input file's name.[6]

4.2.2 Index-only Self-Sort Option

There is a SORT option that can be used under certain conditions to achieve marked savings in the time it takes to self-sort a file, while also conserving disk space. The "IX" (for "index-only") SORT option just rebuilds the index in the existing file, instead of moving the data into another file, and thus saves the time and resources SORT would normally use in creating a temporary file and transferring data into it.

The SORT dialogue for this option is:

```
$ sort
Input file....: input-file-spec
Output file...: ix [fill_%]
Rebuilding index only. OK? y
```

As with any self-sort the IX option requires that the file can be opened for "exclusive" use (see Section 4.2 “SORT Dialogue”).

The option to rebuild the index can be used in any self-SORT provided that:

1. you are not aggregating the file
2. you do not need to recover space from deleted records in the file (the data portion of the file is not touched!)
3. the file has key fields in its DEF (files that have only ASC or DESC fields, rather than KEY or DKEY fields, do not have an internal index).

NOTE

Files that are repeatedly self-sorted with the IX option can get increasingly "out of sequence", i.e. the physical order of the records in the file becomes less and less related to the indexed, key value order of the records. In extreme cases, i.e. if large numbers of records are added to the file without doing a normal, "complete" self-sort, this can seriously affect the performance of ADMINS commands that are optimized for processing files whose records are physically in sequence (such as MAINT, MOVE, the DETAIL file of PROD, REPORT, etc.).

4. A name in the form SORTxx.TMP is automatically generated by ADMINS (see Appendix C.1.1 “Differences in Print File and Temporary File Naming”).
5. In an index-only self-sort no OUTPxx.TMP file is created.
6. SORT will build the temporary working file in the directory assigned to the logical name ADM$SRTMP if it is assigned, and self-SORT will build its output file in the directory assigned to the logical name ADM$SRTOUT, if it is assigned. These SORT logical names are normally not needed and should not be assigned. If needed, however, they should be set up by the System Manager and assigned for the user at login.

Keep this issue in mind when using index-only self-sorts. Its a good idea to self-sort files periodically, especially if the number of records added to the file becomes significant. One technique might be to do index-only self-sorts on a daily basis as batches of records are added to a file, and to do normal self-sorts of the file on a weekly basis.

Typical uses are:

1. Sort a file after records have been appended to it, i.e. with MOVE
2. Sort after changing key values have been altered, i.e. with ADED or MAINT
3. Sort after SEQuentializing the file (see Section 13.4 "FILECONVERT - Convert ADMINS datafile attributes")
4. Sort to compress the index, i.e. after large numbers of record deletions and insertions
5. Sort to expand the index, i.e. to rebuild the index with partially full index blocks (see Section 4.2.2.1 "Rebuilding Index with Partially Full Blocks")

4.2.2.1 Rebuilding Index with Partially Full Blocks

The IX SORT option includes a capability to fill the index blocks partially, i.e. from 50% to 100% full, as specified by the user.

Leaving space in the index blocks reduces the number of new index blocks which are created by subsequent insertion, i.e. in TRANS or PROD. This feature can be used to help manage disk space in large files which undergo random insertion. When this option is requested, as in the following example, SORT partially fills the index blocks which contain pointers to records.[7]

```
$ sort
Input file....: vendor.mas
Output file...: ix 60
Rebuilding index only. OK? y
```

Index blocks with pointers to lower levels of index are completely filled, as usual. See Appendix E for a detailed discussion of ADMINS internal file structure.

4.2.2.2 Rebuilding Indices After Batch Processing

Batch commands that may invalidate indices can be instructed to rebuild those indices (using SORT) on normal termination. This is done using the command line switch **-SORT**.

This function is available in both MOVE (see Section 3.2.6 "SORT: Rebuild indexes after records moved") and MAINT (see Section 10.13 "Rebuilding Indices After Batch Processing").

7. In the example the index blocks of VENDOR.MAS which contain pointers to records would be output 60% full. Building index blocks that are less than 100% full make it likely that the file will have to be enlarged to accommodate the rebuilt index. Note that the IX SORT option, if necessary, will try to enlarge the file to accommodate the rebuilt index BEFORE anything else is done, i.e. if the file cannot be enlarged it will not be changed at all.

4.2.3 KEEPTEXT: Self-Sort without TSF, TCF processing

If SORT is called with the "KEEPTEXT" qualifier on the command line, files that have TInn or TXnn fields will be processed much faster, because SORT will skip the reorganization of the TCF and TSF files (see Appendix K.1 "Special Considerations").

```
$SORT/K
Input file....: evaluation.mas
Output file...: <return>
```

Note that "KEEPTEXT" can be abbreviated to "K".

The trade-off for this increase in speed is that "dead" space in the TCF or TSF is not reclaimed. "Dead" space results either from making a TInn field shorter than it was when it was first created (via editing for example), or from deleting records which have non-blank text fields.

4.3 Operation of SORT

Given a usable input file and output file, SORT begins the sorting process. SORT operates in three distinct "passes", but before beginning Pass (1), SORT reads the input and output file definitions,[8] and does some set up and checking.

Pre-Pass (1) Set Up and Check

a. If the SORT is an ordinary self-sort, then SORT first creates ADM$SRTOUT:OUTPxx.TMP to use as the output file. OUTPxx.TMP is created as an identical empty copy of the input file. If the SORT is an index-only self-sort, no temporary output file is created, but SORT checks immediately that sufficient space will be available for the new index structure. If the file needs to expand SORT tries to enlarge it immediately. In these circumstances if automatic file enlargement (see Section 1.8 "Dynamic Data File Expansion") has been disabled or the disk is full SORT will exit immediately.

In self-sorts SORT then skips to pre-pass step (f).

b. Primary names in the input file are compared with primary and secondary names in the output file to set up tables for moving input records into output records. At this point, SORT checks to see if the input and output fields of the same name have the same **type.** (As with MOVE, alphanumeric and pictured fields may have different **lengths** in their input and output records).[9] If the same field name has a different type in input from output, SORT will print a message as follows and terminate.

```
field-name is not the same in both files
```

8. When we say that an ADMINS command, other than DEFINE, reads a file definition, we do not mean that the instruction file XXXXXX.DEF is read. We mean the command reads the internal copy of the file definition stored in the ADMINS data file itself.
9. SORT requires that the output file key fields match the corresponding input file fields in type and length.

c. SORT checks to see if the output file has a SELECT. If so, SORT relocates the references to field names so the SELECT will execute on input records, and notes to use SELECT during Pass (1). (Note that SELECT is not applicable to a self-sort.)

d. SORT checks that output fields do not receive data from more than one input field. If this condition is detected, SORT prints a message as follows and terminates.

```
field-name receives more than one input value
```

One input field may, however, be directed to two or more output fields.

e. If automatic file enlargement has been disabled (see Section 1.8 "Dynamic Data File Expansion") SORT checks that the size of the output file is sufficient to hold all the input records. If the output file is too small, SORT prints a diagnostic message and terminates.

This particular check is only performed if the output file neither contains a SELECT nor is a definition of a derived aggregate file; because in these two special cases SORT expects the output file to be smaller than the input file.

f. In Pass (1) SORT will read the input file, extract the sort keys, sort them in sections, and write the sections into a temporary working file, ADM$SRTMP:SORTxx.TMP. SORT analyzes its requirements for a temporary working file (SORTxx.TMP) and builds the file. If a SORTxx.TMP already exists, but is of insufficient size to meet SORT's requirements, then SORTxx.TMP is deleted. (Normally, SORTxx.TMP is deleted at the end of the sort.) SORTxx.TMP is usually a relatively small file compared to the sizes of the input and output files.

After performing these checks SORT begins Pass (1). Pass (1) operates as follows.

Pass (1)

a. An input record is read into a record buffer. If an end of file is read instead, then SORT proceeds to step (1c).

b. If a SELECT exists in the output file it is evaluated on the record in the input record buffer. If the SELECT evaluates to "false", then SORT goes immediately to step (1a) for the next input record. (Note that SELECT is not applicable to a self-sort.)

A list of key values and record addresses is built up in memory, and as memory is filled, the list is sorted on key value and written out to ADM$SRTMP:SORTxx.TMP, making room for more keys in memory. SORT continues to read records as per step (1a) until it reaches the end of the input file.

c. Pass (1) of SORT proceeds to Pass (2) of SORT.

Pass (2)

Pass (2) of SORT continues automatically after Pass (1) without any user interaction. Pass (2) merges the sorted keys produced from Pass (1), placing a list of keys and input record disk addresses into ADM$SRTMP:SORTxx.TMP. Then Pass (2) calls Pass (3).

In an index-only self-sort the merged keys are written directly to the input file, overwriting the old index structure, and Pass(3) is called only to close the sorted file, delete the SORTxx.TMP file, and exit.

Pass (3)

Pass (3) of SORT also continues automatically after Pass (2) without any user interaction. If SORT is simply sorting (i.e. not aggregating), Pass (3) does more or less what the MOVE command does. Only instead of reading the input file in sequential order, SORT uses the list of sorted output keys and input record physical addresses written by Pass (2) into ADM$SRTMP:SORTxx.TMP to read the input file in the order that the output file records are to be produced.

After all the input records have been read, and appended to the (originally empty) output file, SORT closes both input and output files. If the output file was ADM$SRTOUT:OUTPxx.TMP, i.e. if SORT was being used just to sort an input file back into itself, then the input file is deleted and the output file is renamed to the input file name. The reason for temporarily building a sort output file when a file is being sorted into itself is that if some error condition stops the sort before it is finished, then the sort input file is still intact, and the sort can simply be re-started when the error condition is corrected.

4.4 Deriving Aggregates (Summarizing Sort)

SORT is also capable of deriving aggregates during Pass (3) of the sort. If the output file contains derivation operations (see Section 2.4.4 "Deriving Aggregates") then SORT performs a different set of operations in Pass (3). Rather than appending the input records to the output file as they are read in sorted order as per the list in ADM$SRTMP:SORTxx.TMP, SORT performs the derivation operations in memory, and only appends summary records to the output file.

We recall that the possible operations were subtotaling (/V), counting non-null or all existences per run (/E, /C), averaging (/AVG), maximum and minimum (/MAX, /MIN), first and last value in the run (/FI, /LA), and take another field from the same record selected by the previous operator (/SA). An output definition could contain several of these operations, applied either to one or several input fields. However, each operation is with respect to the same control break, i.e. the sort keys of the output file. Fields in the output record without a derivation operator will have null values. An example of a derived file of aggregates, showing both file definition and output sample, is contained in Section 2.4.4 "Deriving Aggregates".

4.5 SORT Example Creating an Index File

One use of SORT is for building an index file. Consider the following file definition:

```
* PROPERTY.DEF
*
MAS  20000
*
ACCT#    XA99999  KEY1  "account number"
OWNER    A30             "owners name"
STREET   A15             "street name for property"
#STREET  I               "street # for property"
ADDRESS  A30             "address of owner"
```

PROPERTY.MAS records a variety of information about properties, e.g. the address, owner, owners address, etc. This file is keyed by an account number. Suppose we wished to achieve access to the file by the street address for the property. For example, to answer the question: "Who owns the property at 115 Main Street?"

To do this, we must build an index file that lets us access the property record account number via the street name and number of the property. This index file will let us retrieve an account number for a given street address, and that account number can be used to take us directly to the property record for the given street address. The ADMINS tools that allow usage of such linkages are described elsewhere in this manual. (E.g. Cross Reference Screens (see Section 5.7 "Branches") and the LINK paragraph in SCREEN/TRANS (see Section 5.4.1 "LINK Paragraph"), the LINK statement in REPORT (see Section 7.13.4 "LINK Statement")) The creation of the linkage is done by building an index file.

```
* STREET.DEF
*
*  Street Index
*
IDX  20000
*
STREET   A15     KEY1   "street name for property"
#STREET  I       KEY2   "street # for property"
ACCT#    XA99999        "account number"
```

The above DEF for the index is then defined to create the file STREET.IDX. We then use SORT as follows to build the index

```
$ sort
Input File....: property.mas
Output File...: street.idx
16:04:45.24
************************************************************
16:04:49.00 261 records read 8 blocks 1 section(s)
16:04:49.15 261 pointers merged
************************************************************
16:04:53.61 261 records sorted
$
```

There is still the question of how an index is to be kept up to date as the PROPERTY.MAS file is changed, e.g. when new property records are inserted. Aside from the obvious but not always satisfactory solution of rebuilding the index file with SORT after each set of updates, there are index maintenance facilities in the TRANSaction processor. These facilities are described in Section 5.4.3 "INDEX Paragraph".

Chapter 5:AdmScreen: Compiling Screen Forms

AdmScreen, the screen compiler, reads an instruction file containing the description of a screen, checks it, and then compiles the description into object form for use by TRANS, the transaction processor. The screen instruction file includes the format and instructions for using a screen to update, query, and display data. The file type of the screen instruction file is always ".TRS" for "transaction source". The object file created by the AdmScreen command has the same file name as that of the screen instruction file, but the file type is always ".TRO" for "transaction object". For example, if AdmScreen is given the instruction file "TIME.TRS" it will compile an object file called "TIME.TRO". If the logical name ADM$OBJECT is assigned the TRO is placed in the directory ADM$OBJECT, otherwise it is placed in the same directory as the TRS.

5.1 Outline Of The Screen Instruction File (TRS)

This section describes the rules and conventions for writing a screen instruction file acceptable to the screen compiler. Chapter 6: "TRANS: Screen Transactions" describes the operation of TRANS, which controls the screen according to the "transaction object" (TRO) file compiled by AdmScreen.

The description of a specific screen is made up of five components. The screen header line, the field names section, and the screen layout section are required for each screen. The external files section and the branches section are optional.

A single screen instruction file may contain the descriptions of several related screens. For example:

```
SCREEN-NAME1 ...
...
END
SCREEN-NAME2 ...
...
END
SCREEN-NAME3 ...
...
END
...
```

The outline of the screen description is as follows:[1]

```
                        Screen Header Line
SCREEN-NAME FILE-NAME [LOG-NAME]RPS[/n][RMO-NAME] [KEYWORDS]
[VIDEO FIELD-CLASS VIDEO_ATTR[ FIELD-CLASS VIDEO_ATTR ]... ]
[TRANS_ENV TRANS-ENVIRONMENT-FILE-NAME]
[APPMENU]
                    External Files Section

LINK LINK-FILE-NAME [W] [NULL] [=LINK_NAME]       !link
K TRS-KEY-FIELD-NAME                              !paragraph
KC TRS-KEY-FIELD-NAME
C TRS-FIELD-NAME
L LINK-FIELD-NAME [TRS-FIELD-NAME]
END

APPEND APND-FILE-NAME CONDITION-NAME LETTER    !append
TRS-FIELD-NAME [APND-FIELD-NAME]               !paragraph
END

INDEX INDEX-FILE-NAME [NO-NULL]                !index
TRS-FIELD-NAME [INDEX-FIELD-NAME]              !paragraph
END
                    Field Names Section

D FIELD-NAME [PLACEMENT] [%WINDOW] [%VIDEO][%LOOKUP]      !display
DR FIELD-NAME/TYPE [PLACEMENT][%WINDOW][%VIDEO][%LOOKUP]  !local display

E FIELD-NAME [PLACEMENT] [QUERY-NAME][%WINDOW][%VIDEO][%LOOKUP]
    [LOOKUP sub-statements]                                !editable field


ER FIELD-NAME/TYPE[PLACEMENT][QUERY-NAME][%WINDOW]

[%VIDEO][%LOOKUP]

       [LOOKUP sub-statements]                          !local editable


L FIELD-NAME [PLACEMENT] [QUERY-NAME][%VIDEO][%LOOKUP]  !loggable
           [LOOKUP sub-statements]
LR FIELD-NAME [PLACEMENT][QUERY-NAME][%VIDEO][%LOOKUP]  !loggable
           [LOOKUP sub-statements]                      !refreshable


V FIELD-NAME/TYPE [PLACEMENT] EXPRESSION    !virtual

CAPS FIELD-NAME1 FIELD-NAME2 ...          !CAPS statement
CAP1 FIELD-NAME1 FIELD-NAME2 ...          !CAP1 statement
REQUIRE FIELD-NAME1 FIELD-NAME2 ...       !REQUIRE statement
BOX LINE COLUMN #LINES #COLS [VIDEO_CODE] !BOX statement
ALLOW FIELD-NAME1 FIELD-NAME2 ...         !ALLOW statement
NOQUERY FIELD-NAME1 FIELD-NAME2 ...       !NOQUERY statement
COLOR xx FIELDNAME1 FIELDNAME2 ...        !Use custom colors
LABEL [y,x] 'Label text' FONT ##  TEXTCOLOR color
TIMEOUT Keystroke Macro                   !TIMEOUT statement
SELFBRANCH[2] # [keyfield1 [keyfield2 […]]]  !control SELFBRANCH for
                                          !mutiple indexez
TOOLBAR OFF|[NO]STRINGS|BUTTONS btn1,btn2... !toolbar statement
CALENDAR Y X DATEFIELD [WEEKNO[=IFIELD]]  !Calendar statement
C EXPRESSION                              !check statement
CHECK EXPLANATION TEXT
CLF EXPRESSION                            !check (at) NEXT statement
CHECK EXPLANATION TEXT
M FIELD-NAME EXPRESSION                   !message statement
MESSAGE EXPLANATION TEXT
```

1. The %**WINDOW** keyword applies only to the internal text (TInn) (TXnn) and external text field types (see Section 5.16 “Text Fields”).

2.See Section 5.7.4 “SELFBRANCH: Self-branching with Multiple Index Files”

```
                    Screen Layout Section
SCREEN [PLACEMENT COORDINATES]
literal text and field names to be displayed
                    Branches Section
BRANCHES [BRANCH MENU COORDINATES] [BRANCH MENU TEXT]  !switch screens
BRANCH-NAME SCREEN-NAME [BRANCH-FIELDS]
BRANCH EXPLANATION TEXT

END                          !terminate each screen description with an END
```

5.2 AdmScreen Command Dialogue

The following example shows the dialogue of the AdmScreen command. The screen instruction file, TIME.TRS, includes two separate screen descriptions, TIME and EMPL.

```
$ screen
Screen file name:time
TIME read and compiled
EMPL read and compiled
2 Screen(s) compiled in time.rmo
$
```

The screen instruction file name may alternatively be included on the command line, as follows:

```
$ screen time
```

All the .TRS files in a given directory can be compiled using the following "wildcard" syntax:

```
$ screen *.trs
```

5.3 Screen Header Line

The first component of a screen description is the screen header line, which must be one line only,[3] and which specifies the screen's name, the file on which it operates, the number of records to be simultaneously displayed, and optionally specifies the LOG-NAME, the RMO-NAME, and keywords that control the operations allowable on the screen.

The syntax of the screen header line is as follows:

```
SCREEN-NAME FILE-NAME [LOG-NAME] RPS[/n] [RMO-NAME] [KEYWORDS]
```

3. In general, each instruction line in the TRS must be on one line. The only lines which may be continued on another line are those lines containing "expressions", i.e. virtual fields, check statements, and message statements.

SCREEN-NAME The name for this screen description. This SCREEN-NAME identifies this particular screen when used in the branch paragraphs of other screen descriptions.

FILE-NAME The name of the file on which this screen operates. This file is often referred to as the "master file".

LOG-NAME The name of the field log file in which the field change transaction records should be stored. LOG-NAME is optional (as indicated by the brackets in the screen outline) and is used to override the default field log name obtained by appending ".FLG" to the master file name, e.g. "TIME.FLG". Field logging is described in Section 6.5 "Field Logging".

RPS/n The number of records per screen, i.e. the number of records displayed together, using the screen layout that will follow. As indicated by the brackets in the screen outline, the "/n" is optional and is described below.

The valid range of values for RPS is from 1 to the number of lines in the screen display[a]

If RPS is greater than 1, then the user is specifying a **multi-record screen.** This allows more than one record from the master file to be displayed at one time in the same screen. The last line of the screen layout contains the field designators that will repeat for each record in a multi-record screen. Multi-record screens, and the restrictions which apply to them, are described later in Section 5.9 "Multi-Record Screens".

If RPS is greater than 1 and is followed by "/n", e.g. 8/2, then TRANS is instructed that the last "n" lines of the screen layout contain the repeating field designators for each record in a multi-record screen. This syntax permits multi-line multi-record screens.

The value of RPS is also constrained by the limit of 1000 editable fields per screen. The key fields and all other editable fields in each record in a multi-record screen count against this maximum.

RMO-NAME The name of a record maintenance procedure to be called by the screen. The name is of the form "XXX.RMO". As indicated by the brackets in the screen outline, an RMO for the screen is optional. The RMO can be used to perform complex computations or logical processing on the active record, to support record selection, to perform automatic branching, to prepare values for posting into linked files, and for numerous other purposes. The general rules for writing a record maintenance procedure are described in Chapter 8: "Expressions" and Chapter 9: "CMP: The Record Maintenance Compiler". The specific uses of an RMO behind the screen are described in Chapter 15: "Basic RMO Functions with TRANS" and Chapter 16: "Advanced RMO Functions with TRANS".

a. See Section 5.6 "Screen Layout"

5.3.1 Screen Header Line Keywords

The allowable functions for a screen are set by the keywords which are included on the screen header line. The following are examples of the screen header line showing various combinations of keywords:

```
PERSNL [PERSONNEL]PRF.MAS 1 INSERT DELETE APPEND

TREV1 [PAYROLL]TIMEW1.MAS 10 NOMSG AUTOCR PASSW BETTY

BILLS BILL.MAS CLERK1.FLG 1 QUERY

TRANSFER BUDGET.MAS 1 TRANSFER.RMO LFBACK NOMSG
```

All of the keywords applicable to the screen header line are described below.

5.3.1.1 INSERT, DELETE, or APPEND Records

INSERT

If the keyword INSERT is included on the screen header line, the screen may be used to insert new records into the master file. That is, if a record is requested by entering the key(s) of the record and the record does not exist, the user will have the option to insert the record. Also, the screen will respond to the INS keystroke allowing the user to insert another record with the same key(s) in the master file. The keywords INSERT and MATCH (see Section 5.3.1.8 "MATCH: Require Exact Match") are mutually exclusive.

DELETE

If the keyword DELETE is included on the screen header line, the screen may be used to delete records from the master file. That is, the screen will respond to the DEL keystroke. The DELETE keyword also controls the record transfer function, described in Section 6.6 "Record Moving and Searching".

APPEND

If the keyword APPEND is included on the screen header line, the screen may be used to append new records to the end of the master file. That is, the screen will respond to the APND keystroke.

5.3.1.2 NOMSG: Inhibit On-line Messages

NOMSG

If the keyword NOMSG is included on the screen header line, the messages at the bottom of the screen indicating the field name, field contents, and field type of the active field are not displayed. The TOF (top of file) and EOF (end of file) messages, as well as TRANS mode status messages (AP, UP, IN, and ED) in the upper right corner of the screen are also suppressed by the NOMSG keyword.

These message displays can always be activated (or deactivated) with the MSG keystroke.

5.3.1.3 AUTOCR: Automatic Carriage Return

AUTOCR

If the keyword AUTOCR is included on the screen header line, then when the user **completely** fills an editable field on the screen with characters, the characters are to be immediately examined for acceptance **as if** ENTER had been pressed following the characters. For example, if a field called MONTH were allocated two spaces on the screen, then the two typed characters are processed as soon as the second character is typed into the MONTH field.

TRANS neither waits for, nor expects, the ENTER key. The cursor then automatically moves on to the next editable field. AUTOCR requires that the enterable field width on the screen be exactly equal to the actual number of characters to be typed into the field. In the case of small fields, "precise placement" (see Section 5.6.1 "Precise Placement of Fields") is used to achieve this exact match.

When AUTOCR is in effect the user may still type ENTER **before** reaching the end of the enterable field width. AUTOCR is an option that applies to all the editable fields in the screen.

5.3.1.4 TABBING or QUERY: Field Selection Mode

TABBING

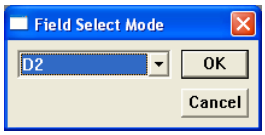
If the keyword TABBING is included on the screen header line, the screen comes up in TRANS using "tabbing" Field Selection Mode and the user can move the cursor from editable field to editable field by pressing the RETURN or ENTER key, or the directional arrows.

The Field Selection Mode can **not** be changed in TRANS from "tabbing" to "query" with the FSM keystroke when the TABBING keyword is used. Hence TABBING is used to restrict a particular screen to "tabbing" only.

Note, if neither TABBING nor QUERY is specified, TRANS uses "tabbing" but **allows** the user to switch back and forth between "tabbing" and "query" using the FSM keystroke.

QUERY

If the keyword QUERY is included on the screen header line, the screen comes up in TRANS using "query" Field Selection Mode. In query field selection mode, the user must select or type the initial letters of a field name (or its query name, if provided in the field names section, see Section 5.5.2 "Editable") to move the cursor to that editable field for data entry.



Query field selection mode is not applicable with multi-record screens.

If QUERY is not included on the screen header line, the screen comes up in TRANS using "tabbing" Field Selection Mode.

If QUERY is included on the screen header line (or if TABBING is not included) the Field Selection Mode can be changed from one mode to the other with the FSM keystroke in TRANS.

5.3.1.5 BREAK On A Multi-Record Screen

BREAK key

If the keyword BREAK followed by a key field name is included on the screen header line of a multi-record screen, the display page will contain only records with the same full or partial key value(s). This feature is described in detail in Section 5.9.2 "BREAK In a Multi-Record Screen".

5.3.1.6 PASSW: Password Protect the Screen

PASSW xxx

If the keyword PASSW followed by a password is included on the screen header line, TRANS will require the user of the screen to provide the password before activating this screen. PASSW should only be used on the screen header line for the first screen description in a screen instruction file, and applies to that particular screen only.

5.3.1.7 Screen Size

132

If the keyword 132 is included on the screen header line, TRANS will create screens that contain 132 characters per line.

LLxWWW

Specify Screens up to 160 characters wide and up to 72 lines long by using the screen size header keyword in the form LLxWWW where LL is the screen length and WWW is the screen width. LL must be a positive integer between 5 and 72, and WWW must be a positive integer between 10 and 160.

Example:

test n.mas 36 showall.rmo 40x140

5.3.1.8 MATCH: Require Exact Match

MATCH

The presence of the MATCH keyword on the screen header line instructs TRANS that when the user requests a record by entering the key value(s), an exact key match between what the user has entered and some record in the master file is required. If an exact match is made, TRANS proceeds to display the desired record. However, if TRANS fails to match on an exact key, TRANS goes into Error Mode. When Error Mode is cleared by the user via the ERR keystroke, TRANS moves to the top of the file and displays the first record.

MATCH also affects screens that are targets of branches (see Section 5.7 "Branches"). If the user branches on a key whose value is not in the target file, MATCH will cause TRANS to display the record at the top of the target file. To request this result, the MATCH keyword is included on the header line of the target screen. MATCH and INSERT are mutually exclusive (see Section 6.6 "Record Moving and Searching").

Section 16.7.1 "Example Using F$F To Secure Student Records" shows an example of the use of MATCH to create a secure screen where a student can only examine his/her own grades in a file of all student grades.

5.3.1.9 SPn or TTn: Print Device Specification

SPn or TTn

Printing directly to a device: The RMO behind a screen can be used to print messages directly to a hard copy printing device by setting the local RMO field P$P (see Section 16.6 "Printing Messages: P$P"). These messages are printed on the device assigned to the logical name ADM$PRT0, unless the screen header line contains either the SPn or TTn keyword. If "SPn" or "TTn" is present, P$P sends the messages to the print queue assigned to the logical name ADM$PRTn. For example, if you want to send messages to the device assigned to ADM$PRT5, then include "SP5" on the screen header line.

5.3.1.10 NOP or SCALE n: Scaling

NOP

This is used to suppress the display of decimal places for non-virtual Dn type fields. When "NOP" is present as a keyword then all non-virtual Dn fields on the screen are displayed as rounded whole numbers.

SCALE n

All non-virtual Dn type fields are to be scaled when displayed on the screen. Scaled values are rounded. The SCALE keyword is followed by a number, n, which represents the power of ten by which the field is to be divided. For example, "SCALE 3" would display each value rounded to the nearest thousand, i.e. "1,234" as "1", "23,642" as "24", and "14,483.45" as "14".

5.3.1.11 NOLOG: Suppress Field Logging

NOLOG

If the keyword NOLOG is included on the screen header line, no log files are to be maintained for the master file when under control of this screen. If NOLOG is absent, then all changes to loggable fields are written into the field log file specified on the screen header line or into the default field log file for the master file if no field log file was specified. NOLOG suppresses the field logging for "L" fields and "LR" fields (see Section 5.5.3 "Loggable"). When NOLOG is present, L fields are treated like E fields and LR fields are treated like ER fields (see Section 5.5.2 "Editable")

5.3.1.12 NOWRITE After Each Field Change

NOWRITE

The active record is normally written back to the master file when any field in the active record is changed by the user manually entering or overwriting a field on the screen. If NOWRITE is included on the screen header line, then the active record is not written back to the disk after each manual field change, but rather the active record is written only whenever the RMO requests it. This could be after all changes have been made to the active record. NOWRITE requires an RMO behind the screen to write updates to the disk, and is described fully in Section 16.1.1 "High Volume Update: NOWRITE". NOWRITE only applies to Update Mode. (In Insert and Append Modes, fields are always written back to the disk by pressing NEXT.)

NOWRITE is incompatible with LFEXIT control, whether explicitly specified with the LFEXIT or LFBACK keyword (see Section 5.3.1.18 "LFEXIT or LFBACK: Update Mode Control"), or implicitly requested via the REQUIRE statement (see Section 5.5.5 "REQUIRE Statement").

5.3.1.13 PREV, NEXT: Record to Display if Key not Found

PREV and NEXT

When a partial or non-existent key value is entered in update mode, TRANS, by default, displays the last record whose key value is less than the key value that was entered. If "k" (lowercase) is included in the string assigned to the logical name option (see Appendix A: "Options"), TRANS' default action in this circumstance is changed so that the record displayed is the first record whose key value is greater than the key value that was entered.

The keywords PREV and NEXT are used to modify the default action for the current screen, so that when a partial or non-existent key is entered, TRANS will always display the "previous" record, if PREV is present in the screen header line, or always display the "next" record, if NEXT is present in the screen header line, irrespective of the option "k" setting.

For example, to find names beginning with "C" in a name index a user enters "C", but TRANS displays the last "B" name instead of the first "C" name. If NEXT is on the screen header line then TRANS will always display the first "C" record, whether or not OPTION "k" is in effect.

5.3.1.14 NOBR: Inhibit Manual Branching

NOBR

If the application designer wishes to control all branching via automatic branching,[a] manual branching can be inhibited with the NOBR keyword. Manual branching (i.e. via the BRNC keystroke) will not be allowed from the screen even though the screen does contain a BRANCHES paragraph.

a. see Section 16.2 "Automatic Branching: B$B and R$R"

5.3.1.15 NOXR: Prevent Return to Screen by Browsing Keys

NOXR

The NOXR keyword prevents the user from returning to the screen via the XRET or XFWD keystrokes. Ordinarily these keystrokes allow the user to browse back and forth through the screens that have been visited in the current TRANS session. A screen that has the NOXR keyword present is simply not included in the "stack" the supports this browsing capability.

5.3.1.16 NOEX: Inhibit Screen Exit

NOEX

The NOEX keyword prevents the user from exiting the screen via the EXIT keystroke. This feature is usually used in conjunction with other related features (automatic branching, see Section 16.2 "Automatic Branching: B$B and R$R", and automatic exit from TRANS, see Section 16.2.4 "Automatic Exit From TRANS: B$B = 'CB'") to cause the user to exit TRANS via a prescribed method.

5.3.1.17 NOTR: Inhibit Manual TRANS Entry

NOTR

If the application designer wishes to access a TRO file only via a branch from another TRO file, TRANS can be inhibited from activating a screen directly by placing the NOTR keyword on the screen header line of the first screen description in the TRS.

5.3.1.18 LFEXIT or LFBACK: Update Mode Control

LFEXIT

The LFEXIT keyword increases control of data entry in Update Mode. LFEXIT prevents the user in Update Mode from filing a record, into which at least one non-key field has been entered, by any means other than pressing NEXT.[a] LFEXIT control is activated when the user enters or alters data in any non-key field in the screen in Update Mode. If the user tries to leave the record by any means other than the NEXT key, TRANS gives the message "LFEXIT ACTIVE, USE NEXT TO FILE RECORD". TRANS does not call the RMO, or write the record to the disk, or go to the next record. The user must press the ERR key to clear the error condition and then must press NEXT to file the record.

When LFEXIT control is active, TRANS will not branch to another screen. However, because TRANS HELP can be obtained by pressing BRNC H (an alternative to the HELP keystroke), if the user presses the BRNC key in Append Mode, Insert Mode, General Editor Mode, or in Update Mode when LFEXIT is active, TRANS prompts "PRESS H FOR HELP" rather than "BRANCH TO".

LFEXIT control represents an alternate mutually exclusive functionality to the NOWRITE screen header keyword. Therefore, the keywords LFEXIT and LFBACK, as well as the REQUIRE statement, are considered syntactically incompatible with NOWRITE. Also, when LFEXIT control is active, setting the W$W field is usually unnecessary (see Section 16.1 "Controlling Changes Written To Disk")

LFBACK

The LFBACK keyword indicates LFEXIT control in Update Mode and also enables a "backout" key, PREV, to allow the user to leave a record intact in a screen in which LFEXIT control has been activated. When LFBACK is specified, and the user has entered or altered fields on the screen but cannot or does not want to complete entry into the record via NEXT, the user can press PREV to back out of the record. This causes TRANS to prompt for confirmation in a pop-up box. The user must click OK to confirm the back out of the changes to the record. Then TRANS restores the original values in the record, and returns to the beginning of the record. (The EOFREC RMO call does NOT occur; the BEGREC RMO calls occur; links are re-executed; and the cursor goes to the first editable field. See Section 15.1 "Communication with TRANS".)

LFBACK is not a way for the user to circumvent the validation logic in a screen. When the user backs out of a record the changes are **not** written back to the disk. Instead, LFBACK simply provides an escape hatch to prevent users from becoming locked into a record which they are unable or unwilling to complete.

See Section 6.3.1.1 "Update Mode Under LFEXIT Control" for additional information on Update Mode under LFEXIT control, including a discussion on field logging with LFEXIT.

a. The RMO behind the screen can simulate the user pressing NEXT, by setting the local field B$B to 'LF'. Other uses of B$B are ignored when LFEXIT control is active. (See Section 16.2 "Automatic Branching: B$B and R$R".)

5.3.1.19 SHORT: Conserve MD Array Space

SHORT

In very large screens, MD ("meta data") array space can be conserved by using the screen header line keyword SHORT. SHORT prevents TRANS from storing the names of fields in LINK files in the MD array unless the fields are referenced in the TRS. SHORT cannot be used in conjunction with the following subroutines which need access to all field names in an external file

MOVFLD, see Appendix H.13.15 "MOVFLD - Move Fields Among Files Accessed via TRO"

FNDTAB, see Appendix H.10.3 "FNDTAB - Set Up Data for LODTAB"

LODTAB, see Appendix H.10.4 "LODTAB - Load Data Into An Array Based On FNDTAB"

SHORT can be specified for individual LINK paragraphs, to conserve MD array space in screens where SHORT cannot be applied to all LINKs (see Section 5.4.1 "LINK Paragraph").

5.3.1.20 COMMA/NOCOMMA

COMMA
NOCOMMA

Controls whether AdmTrans displays thousands separators for numerical fields at the screen level.

COMMA inserts thousands separators (e.g. 12,345.77)

NOCOMMA suppresses the thousands separators (e.g. 12345.77)

If no COMMA or NOCOMMA keywords are present the insertion of thousands separators is governed by the presence or absence of "," (comma) in the string assigned to the logical name OPTION (described in Section 2.4.2.1 "Input and Output Representation Options").

5.3.1.21 NOTMO

NOTMO

Prevents a screen from timing out because of the global timeout feature (see Section 6.17.7 "Global Timeout"). Normally this should only be used for a timeout holding screen to prevent that screen from repeatedly branching to itself.

5.3.2 TRANS_ENV Statement

A statement of the form:

```
TRANS_ENV pathname
```

can be included in the TRS to specify a particular TRANS Environment File (see Section 6.17 "The TRANS Environment File") to use for all the screens in this TRO.[4]

The file named in the TRANS_ENV statement overrides the setting of the logical name TRANS$ENV. The TRAN_ENV statement should appear between the screen header line and the external file section.

5.3.3 ADM_DD Statement

A statement of the form:

```
ADM_DD pathname
```

can be included in the first screen of the TRS to specify an alternative ADMINS Data Dictionary (see Appendix I: "ADD: The ADMINS Data Dictionary") to use for all the screens in this TRO.

The Data Dictionary files in the folder specified in the ADM_DD statement will be loaded and used if the specified path resolves to a different location than is specified by the logical name ADM$DD. The ADM_DD statement should appear between the screen header line and the external file section.

5.3.4 APPMENU Statement

Use the APPMENU statement to activate the AppMenu subsystem for a screen. Use the AppMenu subsystem to create application specific menu bars and control the execution of command files and reports, as well as verifying parameters necessary to run the reports and command files. See Appendix M: "The AppMenu SubSystem" for details.

The APPMENU statement should appear between the screen header line and the external file section.

5.4 External Files

The second component of the screen description, external files, is concerned with the relationship of the active record on the screen to records in other (external) files. There are many kinds of operations that can be specified in the external files section: **linking** to existing records in another file, **appending, inserting, or deleting** records in another file to reflect user actions, and maintenance of another file as an **index** to records in the master file.

5.4.1 LINK Paragraph

The LINK paragraph is used to include fields from files other than the screen's main file in the active (virtual) record being displayed. Each LINK paragraph names the link file, the fields from the active record that form a key into the link file, and the

---

4.Only one TRANS_ENV statement is allowed per TRS. If you want another screen to utilize a different TRANS_ENV file, you must place that screen in another TRS.

names of the fields in the link file to be included in the screen's virtual record. These fields to be used through the link may be called by their link file names or may be "renamed" in the virtual record. The syntax of a LINK paragraph is as follows:

```
LINK LINK-FILE-NAME [W] [SHORT] [NULL] [=LINK_NAME]
K   TRS-KEY-FIELD-NAME
KC  TRS-KEY-FIELD-NAME
C   TRS-FIELD-NAME
...
L   LINK-FIELD-NAME [TRS-FIELD-NAME]
L   ...
...
END
```

LINK-FILE-NAME is the file that contains the records to which the linkage is made. The "W", which is optional, means that the screen may be used to alter fields in the link file. Linked fields that are altered in the virtual record are usually "written back" when TRANS leaves the currently active virtual record.[5]

If necessary, link writing can be controlled more precisely via a record maintenance procedure running with TRANS, as described in Section 16.1 "Controlling Changes Written To Disk".

As is described in Section 5.3.1.19 "SHORT: Conserve MD Array Space", the TRS header line keyword SHORT prevents TRANS from storing the names of fields in LINK files in the DA array unless the fields are referenced in the TRS. This saves DA array space which can become scarce in complex screens.

When it is not be possible to use SHORT for an entire screen, you can specify SHORT for individual LINK paragraphs. As with SHORT on the TRS header line, SHORT must not be used when the LINK file is used with the MOVFLD, FNDTAB, or LODTAB subroutines.

NULL is an optional keyword that tells TRANS to try the link even if the key values it has to make the link are all null (i.e. blank for alphas, zero for numerics). **NOTE: Unless NULL is the last item on the first line of the LINK paragraph, or immediately before the "=LINK_NAME" link field renaming string, TRANS does not try or retry a link when the key values it has to search for are all null.**

=LINK_NAME[6] is an optional syntax for automatically renaming all the fields linked in by a LINK paragraph. For example:

```
LINK COURSE.TAB W =CRS_
KC CID
L INSTRUCTOR
L CNAME
```

will automatically rename all the fields linked by the LINK paragraph by prefixing "CRS_" to the name of the field in the link file (i.e. linked in fields named "INSTRUCTOR" and "CNAME" would be renamed to "CRS_INSTRUCTOR" and "CRS_CNAME" for the remainder of the screen). =LINK_NAME must be the last item on the first line of the LINK paragraph, or immediately before the NULL keyword.

5. By default in single-record screens links are written back when the active record is cleared from the screen, i.e. when the user presses the NEXT keystroke, or at other end-of-record processing points as described in Section 15.2.3 "End of Record Processing: S$S = 'EOFREC'". Multi-record screens, by default, write back links whenever TRANS leaves the currently active record (e.g. via UP or DOWN arrows, or the PREV, NEXT, EXIT, BRNC, HOME, or NREC keys, etc.)
6. This same syntax is used to identify particular links for the ADM$NLREC special RMO array, which is used to identify which files in a screen are currently ignoring record lockout, as described in Section 16.22.2 "Multi-Record Summary Screens". If you wish to utilize ADM$NLREC **without** automatically renaming the link fields, use a dash (-) in place of the equal sign (=) in the syntax, i.e. -LINK_NAME.

K in column 1 means that the named field (TRS-KEY-FIELD-NAME) is to be used as a key into the link file.

KC means the named field (TRS-KEY-FIELD-NAME) is to be used as a key, and that the link fields should be fetched whenever this particular field is changed on the screen.

C means that the link fields should be fetched whenever this particular field (TRS-FIELD-NAME) is changed[7] on the screen.

The TRS-KEY-FIELD-NAMES which are used as keys to match a record in the link file must be presented in the LINK paragraph in the order in which the keys are defined in the link file.

Next the field names in the linked record (LINK-FIELD-NAME) that are to be obtained via the linkage are listed. These field names are presented on "L" lines in terms of the name used in the link file, but they may be optionally renamed (TRS-FIELD-NAME) for use in the remainder of the screen description. (You must rename the link fields if a field with the same name already exists in the active record).[8]

Both the field(s) used as the key(s) for the link, and the linked-in fields themselves, are restated in the field names section if they are to be included in the screen display. They may be D (display only) or E (editable) fields. The cursor movement sequence between editable fields is determined by their position in the field names section of the screen description. (Fields used as the keys in the LINK paragraph may also be L (loggable) fields or they may be "DR" or "ER" "local" fields. See Section 15.1.3 "Local Fields in the RMO" for an explanation of the "local" fields in the RMO.)

There may be many LINK paragraphs in a screen description. Also there may be several LINK paragraphs to the same external file in the same screen description. (Several LINK statements invoking the same file name cause that file to be opened only once by TRANS.)

Let us look at an example. The active record concerns a student registering for a course, i.e. there are fields called STUDENT and COURSE. There is also a master course file that contains descriptive information about each course, such as the course name (CNAME), the instructor (INSTRUCTOR), the meeting place (PLACE), the number of students (NSTDTS), etc. The LINK paragraph relating the student record (i.e. the active record) to the course record would look as follows.

```
LINK COURSE.TAB W
KC CID
L  CNAME
L  INSTRUCTOR TEACHER
L  PLACE
L  NSTDTS
END
```

7. TRANS can alternatively re-read linked data directly from the disk whenever a KC or C field is entered, whether or not the value in the field is actually changed. This feature also overrides the buffering of linked data that is normally done in TRANS, and ensures that TRANS has access to absolutely up-to-date data as it exists on disk. This is enabled by including "s" (lowercase) in the string assigned to the logical name OPTION (see Appendix A: "Options").
8. All the fields linked by a LINK paragraph can be renamed automatically via the "=LINK_NAME" syntax on the first line of the LINK paragraph, as described earlier in this section.

The name of the master course file is COURSE.TAB. The linked course record may be modified by the user and consequently the course records are written back to the course master file when a student record is cleared from the screen, i.e. at end-of-record processing (explained more fully in Section 15.2.3 “End of Record Processing: S$S = 'EOFREC'”). This is signified by the "W" following the course file name.

The course id (CID) field is used to form the link. Whenever a student record appears on the screen with a **non-null** CID value, or whenever the user changes the value in the CID field, then the student record is linked to the course record, and the fields called CNAME, INSTRUCTOR, PLACE and NSTDTS, are filled in on the screen. In the sample screen description containing the LINK paragraph above, the field called INSTRUCTOR in COURSE.TAB is renamed TEACHER in the screen.

TRANS treats errors entered into KC fields in a special way. When an entry error is detected TRANS enters Error Mode, which is cleared by pressing the ERR keystroke (see Section 6.3.4 “Error Mode”). Then the original value of the field whose entry triggered the error condition is redisplayed. If, however, the field triggering the error was a KC field in a LINK paragraph, then **all** the key fields (K and KC) in the LINK paragraph are reset to the original values that were in those fields before the link was executed. (If the logical name OPTION (see Appendix A: “Options”) includes the letter "L", only the actual field being entered is reset and the other K and KC fields are not reset.)

5.4.1.1 Chain Linking

It is possible that a linked field in one link paragraph is the key field in a subsequent link paragraph. That is, the second link should be re-executed whenever the KC field in the first link paragraph is changed. This is requested by placing the letter "C" (for link when **changes)** along with the KC **(key** and link when **changes)** fields in the second dependent link paragraph. The "C" next to a field name in a link paragraph causes reevaluation of the pertinent link when the field changes, even though the "C" field is not part of the key in the link being reevaluated. For example:

```
LINK ADDRESS.TAB
KC SS#                  "social security number"
L STREET                "link the address and ZIP for a SS#"
L ZIP
END
*
LINK ZIP.TAB
KC ZIP                  "ZIP is the key to link the town name"
C SS#                   "retry this link when SS# is changed"
L TOWN
END
```

5.4.1.2 Linking Without an Exact Match

The LINK function will either find an exact match in the link file or, if no match is found, will return with null values for the link fields. Four alternative linkage operations are also available in situations when an exact match may not be found but when an actual link is desired. These operations compare the link key values to the key values in the link file and link to the next higher or lower record in the link file, when there is no exact match, or even if there is an exact match.

1. **LINKGT** - Link Greater Than: Links to the next higher record in the link file even if there is an exact match. If there is none higher, null values are returned for the link fields. This happens when the link key values are equal to or exceed the last record in the link file.

2. **LINKGE** - Link Greater than or Equal to: Links to an exact match, or if one is not found, links to the next higher record in the link file. If there is none higher, null values are returned for the link fields. This happens when the link key values exceed the last record in the link file.
3. **LINKLT** - Link Less Than: Links to the next lower record in the link file even if there is an exact match. If there is none lower, null values are returned for the link fields. This happens when the link key values are lower than or equal to the first record of the link file.
4. **LINKLE** - Link Less than or Equal to: Links to an exact match, or if one is not found, links to the next lower record in the link file. If there is none lower, null values are returned for the link fields. This happens when the link key values are lower than the first record of the link file.

In this withholding table example, a SALARY is compared to a LO amount to determine a base tax amount plus a percentage to be applied to the difference between the SALARY and the LO amount.

```
*    WITHOLD.DEF
TAB 100
MARITL A1 KEY1      "Marital Status"
LO D2 KEY2          "Low Side of Salary Range"
AMT D2              "Base Tax Amount for Low"
PERCENT D2          "Percent Applied to Difference Over Low"
```

A link to a record in this table with an exact match or less than an exact match extracts the suitable value for computing the withholding.

```
...
LINKLE WITHOLD.TAB
KC MARITL
KC SALARY
L AMT
L PERCENT
END
...
```

The withholding table includes the following data:

```
MARITL          LO       AMT       PERCENT
...
S            4,000       150          0.12
S            6,000       275          0.14
S            8,000       400          0.17
...
```

If the link keys are "S" and "6,400", the linked fields AMT and PERCENT will contain the values "275" and "0.14" respectively. Note that if a LINK statement was used instead of the LINKLE statement, null values would have been returned for the link fields because an exact match of key values is not found in the link file.

5.4.2 APPEND Paragraph

The APPEND paragraph is used to instruct TRANS to insert, append or delete a record in an external file using fields from the active record [9]The active record consists of actual fields or virtual fields or fields calculated by a record maintenance procedure (RMO) operating behind the screen. (The APPEND paragraph should not be confused with the APPEND keyword in the screen header line which enables Append Mode described in Section 6.3.2 "Append Mode". The APPEND paragraph pertains to adding records to an external file, whereas Append Mode pertains to adding records to the master file.)

The syntax of the APPEND paragraph is as follows:

```
APPEND APND-FILE-NAME CONDITION-NAME LETTER
TRS-FIELD-NAME [APND-FIELD-NAME]
TRS-FIELD-NAME [APND-FIELD-NAME]
...
END
```

The paragraph instructs TRANS that when the user at the terminal (or the RMO) sets the field called CONDITION-NAME to certain specific values then certain specific actions should occur. There are three condition letters reserved for special functions, which therefore should **not** be used to instruct appends.

Condition Letter	Description
I	Insert whether or not this key is already present.
A	Add, i.e. insert only if this is a new record.
D	Delete the record with this key.

All letters other than I, A or D instruct appending, i.e. adding the record to the end of the file.

When records are inserted or appended, then those fields in the active record called TRS-FIELD-NAME should be used to set the fields called APND-FIELD-NAME to make a new record for the APND-FILE-NAME file. Then TRANS resets the CONDITION-NAME field to blank. The CONDITION-NAME field, which may be an actual field in the file or a local ER (or DR) field, must have the field type A1.

(As with LINK, the active fields (TRS-FIELD-NAME) in the APPEND paragraph may be virtual fields (see Section 5.5.4 "Virtual Fields") or local fields in a record maintenance procedure (RMO) behind the screen (see Section 15.1.3 "Local Fields in the RMO"). If any of these fields are local in an RMO they must also appear in the field names section of the screen.)

For example, the following APPEND paragraph can be used to add or delete a record to a payment file (PAYM.MAS) from a purchase order screen running on a purchase order master file. When the condition name ACTION is set to "I", then the fields #VEND, PDATE, #INVOICE, AMOUNT, #PO from the purchase order master file are inserted as a record to PAYM.MAS.

9. Generally the APPEND paragraph can add records to any ADMINS data file, including the main file of the screen. However, if the screen is a multi-record screen (see Section 5.9 "Multi-Record Screens"), the APPEND paragraph file cannot be the main file of the screen.

```
APPEND PAYM.MAS ACTION P
#VEND
PDATE
#INVOICE #INV
AMOUNT AMT
#PO
END
```

Notice that the fields in the purchase order master file record called #INVOICE and AMOUNT are called #INV and AMT in PAYM.MAS.

When the user enters the letter "I" in the field "ACTION", the record is added to the external file immediately, and the CONDITION-NAME field (ACTION) is cleared. Thus several records can be added to the external file from one active record on the screen. (Section 15.4.2 "Example of Appending Via the RMO" describes a more automatic method of appending records to external files.)

5.4.3 INDEX Paragraph

ADMINS data files maintain an internal index of key values that allows records to be accessed directly (see Appendix E: "File Concepts"). Finding a record directly (without doing a sequential record-by-record search) using any other combination of fields requires an **external index** to the file. An external index must be built and maintained for each additional index criterion (combination of fields) by which you want direct access to information in the master file. External indexes provide quick access to the data by various index criteria while the bulk of the data is kept in a single file.

To illustrate:

A real estate tax assessment file is ordered as follows:

```
*
MAP         X99       KEY1      "Map number"
PARCEL      X9999     KEY2      "Parcel number"
UNIT        A10       KEY3      "Unit ID for multi-unit"
*
```

Each record contains information about a property, including the information about who owns the property:

```
*
OLASTNAME   A20                 "Owner's last name"
OFIRSTNAME  A20                 "Owner's first name"
OMI         A20                 "Owner's middle init."
*
```

Records in the assessment file can be directly accessed via the key fields, i.e. in TRANS you can enter MAP, PARCEL, and UNIT to find a record. If, however, you want to find assessment data using the property owner's name, you can build an **index file** that cross-references the MAP/PARCEL/UNIT combination for each owner's name (OLASTNAME/OFIRSTNAME/OMI). Screens (and reports, see Chapter 7: "AdmREPORT: Creating Reports") can then use the owner's name to look up MAP/PARCEL/UNIT in the index file, and then LINK using that value to the assessment file.

External index files are simply ADMINS files created with DEFINE. The fields you want indexed are keys in the index file. The key fields of the master file are data fields in the index file. The index file is loaded by SORTing the records of the master file into the index file (see Section 4.5 "SORT Example Creating an Index File").

The file definition for the index file in our above illustration may look as follows:

```
IDX  10000
LAST        A20       KEY1      "Owner's last name"
FIRST       A20       KEY2      "Owner's first name"
MI          A20       KEY3      "Owner's middle init."
MAP         X99                 "Map number"
PARCEL      X9999               "Parcel number"
UNIT        A10                 "Unit ID for multi-unit"
```

In screens, the INDEX paragraph maintains an external index file for the master file by inserting and/or deleting records in the index file when the values of the non-key fields being indexed are entered or changed in the main file.

Continuing the above property tax assessment illustration, if a property changed hands, the MAP/PARCEL/UNIT record for that property might be edited in a screen, with the new owner's name substituted for the old. An INDEX paragraph would automatically delete the old owner's record from the index file, and insert a new record for the new owner of that MAP/PARCEL/UNIT. The INDEX paragraph maintains the index file as if it had just been sorted from the master file.

The INDEX paragraph has the following format:

```
INDEX          INDEX-FILE-NAME      [NO_NULL]
TRS-FIELD-NAME [INDEX-FIELD-NAME]
TRS-FIELD-NAME [INDEX-FIELD-NAME]
...
END
```

After the INDEX keyword, the index file is named (INDEX-FILE-NAME). Then the fields in the active record that are the index criteria (the keys in the index file) are named, followed by the key fields in the active record (TRS-FIELD-NAME). The fields must be fields in the active record, not local fields or link fields in the screen. When the TRS-FIELD-NAME for a field differs from the name of the corresponding field in the index file (INDEX-FIELD-NAME), both names are stated.

The index paragraph and field names section for a screen that displays property assessment records might be as follows:

```
...
INDEX OWNER.IDX
OLASTNAME LAST
OFIRSTNAME FIRST
OMI MI
MAP
PARCEL
UNIT
END
...
E MAP
E PARCEL
E UNIT
...
E OLASTNAME
E OFIRSTNAME
E OMI
...
```

Note that in the INDEX paragraph, both the field name in the master file and the field name in the index file ("OLASTNAME LAST") are given when they are different.

When the user clears a tax assessment record from the screen, TRANS will check if any index criterion field has been altered, namely OLASTNAME, OFIRSTNAME, or OMI. If so, TRANS automatically updates the index file.

If a record is deleted from the master file, TRANS updates the index by deleting the corresponding index record. If a record is added to the master file, TRANS inserts a corresponding index record in the index file.

Index files are updated during "end of record" processing, as described in Section 15.2.3 “End of Record Processing: S$S = 'EOFREC'”.

5.4.3.1 NO_NULL: Suppress Null Keys in Index File

The NO_NULL keyword after the filename on the first line of the INDEX paragraph instructs TRANS not to insert records in the index file with null key values.

```
INDEX FILE-NAME NO_NULL
```

In the example from the previous section, if the master file record for a MAP/PARCEL/UNIT was updated and the fields for the owner's name were all changed to blank, the INDEX paragraph would normally delete the record for the old owner in the index file and insert a new record with null (blank) keys. **With NO_NULL in effect records with null keys are not inserted in the index file,** If NO_NULL was in effect, the INDEX paragraph would remove the record for the old owner but would not insert a record with a null key.

NO_NULL also prevents records with null keys from being inserted into the index file when new records are added to the master file that have null values for indexed fields.

5.5 Field Names

The third component of a screen description, the field names section, identifies the fields to be displayed or referenced by the screen. The field names section may also contain:

1. Message statements to be displayed on the screen
2. Check statements for validating data entry
3. REQUIRE statements which set out the required fields
4. BOX statements which define graphic display boxes and lines
5. ALLOW statements that identify fields for special entry processing
6. NOQUERY statements that exclude fields from Query Mode processing.
7. CAPS (case-insensitive entry) and CAP1 (capitalize each word) statements.
8. PushButtons

The order of the fields in the field names section determines in what order the cursor will move from field to field. The field names are placed one to a line.

Non-key fields in the master file which are not to be displayed need not be included in the list, but all the key fields must appear in order, together, in the list even if they are not actually displayed on the screen.

The name of each field is preceded by a code which controls the role of that field name on this particular screen.

These codes are as follows:

5.5.1 Display

D

D stands for display. D fields are only displayed, and cannot be changed via this particular screen. D fields are fields from the master file or fields that have been linked via a LINK paragraph. The cursor skips over D fields when the user presses a directional arrow or the ENTER keystroke, unless the field is a text field or has a window.[a] For example:

```
D NAME
D PAYCODE
```

DR

DR also stands for display; DR fields cannot be changed directly by the user at the terminal. The cursor skips over these fields. The "DR" designation is usually used for fields which are "local", i.e. are not in an actual record, and are usually associated with a record maintenance procedure (RMO) behind the screen. The DR designation may also be used on actual fields that will be changed by the RMO, and need to be refreshed (redisplayed) by TRANS. A more complete discussion of "local" fields is in Section 15.1.3 "Local Fields in the RMO". For DR fields which are **local,** the syntax must include the field type. For example:

```
DR CODE/A2
DR AMOUNT/D2
DR COUNT/I
```

For DR fields which have already been encountered in the TRS by AdmScreen (i.e. fields in the master file or in link files), the field type specification is optional. For example:

```
DR ENCUMB
DR BALANCE
```

When the field type is included for DR fields already encountered in the master file or link file, AdmScreen verifies that the specified field type matches the field type defined in the file, and exits with an error message in case of a field type mismatch.

A reference to a Data Dictionary element may be substituted for the field type specification as described in Section 1.3.5 "Referencing Data Dictionary Elements", e.g.

```
DR LASTPO#/@PO#
```

a. The cursor stops at a D field if it is a text field (field types TInn and TXnn) so that you can view (only) its contents. See Section 5.16 "Text Fields". The cursor will also stop at a D field if a LOOKUP window is specified for that field, so that you can display the window. See Section 5.11 "LOOKUP Window".

5.5.1.1 Restrict TRANS to Key Range

DL

DL stands for Display and Lock. DL fields restrict TRANS to a key range, starting with the highest key, KEY1. If the screen contains "DL KEY1_FIELD", then KEY1_FIELD will not be editable, and need not even appear on the screen. If TRANS branches to this screen and KEY1_FIELD has a value of 500, then TRANS will only access records where KEY1_FIELD is 500.[a]

TRANS will not allow deletion of the last record in a locked range. Record transfer operations (the TRF key), and record insertions will only insert records in the locked range, as only the unlocked keys are prompted for. Append mode is not allowed on screens that have DL fields. If APPEND is present on the screen header line, a warning message is issued and the APPEND keyword is ignored.

The following field declarations would restrict TRANS to only those records with the values of FUND (KEY1), DEPT (KEY2), and OBJ (KEY3) used to enter the screen.

```
DL FUND
DL DEPT
DL OBJ
```

If a screen utilizes DLonly alternate indices that have the exact same fields from key 1 down to and including the DL field will appear in the File/Alternate Indices menu (this restriction assures that a user cannot access records outside the locked key range by switching to another index).

a. It is the responsibility of the application developer to ensure that a branch is made to the intended range. DL locks TRANS to the key range defined by the record it lands on after the branch, which will be the NEXT record in the file after the branch key values specified, if no record matches the branch key values.

5.5.2 Editable

E

E stands for editable. E fields from the master file can be changed on the screen by the user. Changes to E fields are not logged into the field log (see Section 5.3 “Screen Header Line”). Rather, L for loggable, should be used if logging is desired.

The field name can be followed by a "query" name. The query name, if present, is used to perform the initial letter matching search during query mode described in Section 5.3.1.4 “TABBING or QUERY: Field Selection Mode”; a match on the query name moves the cursor to the field to be entered. Otherwise the name from the file definition is used for initial letter matching. For example:

```
E NAME
E NAME LASTNAME
```

If the editable field is a text field (field types TXnn and TInn) you must open the TED window on the file in order to edit it, as described in Section 5.16 “Text Fields”.

ER

ER also stands for editable. The relationship of an ER field to an E field is the same as described above for a DR versus a D field. For **local** ER fields the syntax must include the field type and may have a query name (see Section 5.3.1.4 "TABBING or QUERY: Field Selection Mode").

ER fields are typically used for editable local fields displayed on the screen, actual fields in the file which are change by the RMO behind the screen (see Section 15.1.3 "Local Fields in the RMO") and need to be refreshed by TRANS, editable link keys or link fields which are either local fields or are being changed by the RMO. For example:

```
ER CODE/A2
ER AMOUNT/D2 AMT
ER COUNT/I
```

For ER fields which have already been encountered in the TRS by AdmScreen (i.e. fields from the master file or link files), the field type specification is optional. For example:

```
ER ENCUMB
ER BALANCE
```

When the field type is included for ER fields already encountered in the master file or link file, AdmScreen verifies that the specified field type matches the field type defined in the file, and exits with an error message in the case of a field type mismatch.

A reference to a Data Dictionary element may be substituted for the field type specification as described in Section 1.3.5 "Referencing Data Dictionary Elements", e.g.

```
ER LASTPO#/@PO#
```

5.5.3 Loggable

L

L stands for loggable. L fields are fields from the master file that operate as E fields, except that changes to L fields are logged in the field log file. L fields may have a query name. For example:

```
L HOURS
L PAYRATE RATE
```

LR

LR stands for loggable and refreshable. The LR designation allows editable fields from the master file to be changed by the RMO and refreshed on the screen, and logged in the field log file. LR fields operate only when LFEXIT control is active (see Section 5.3.1.18 "LFEXIT or LFBACK: Update Mode Control"). That is LR fields operate when the keyword LFEXIT or LFBACK is included in the screen header line, and/or there is at least one REQUIRE statement (see Section 5.5.5 "REQUIRE Statement").

The syntax for LR fields may include a query name. The field type specification is optional. When the field type is included, AdmScreen verifies that the field type matches the field type in the master file and exits with an error message in the case of a field type mismatch.

```
LR HOURS
LR PAYRATE RATE
LR TOTAMT/D2
```

A reference to a Data Dictionary element may be substituted for the field type specification as described in Section 1.3.5 "Referencing Data Dictionary Elements", e.g.

```
LR LASTPO#/@PO#
```

5.5.4 Virtual Fields

V

V stands for virtual. V fields do not exist in the data record, whereas D, E and L field names are all from the data record. Rather V fields are computed for each record for display only. For example:

```
V XFUND/X9999 0100
V MSG/A7 'Overdue'
V TAXAMT/D2 6.75
V GROSS/D HOUSE + OUTBLDG + LOT + FARM
V RETIREMENT/D IF GROSS - PAY LE 6600 :
THEN GROSS - PAY * .025 :
ELSE GROSS - PAY * .05 END
```

The rules for computing V fields are identical to the rules for the CREATE statement in AdmREPORT. These rules are discussed in Chapter 8: "Expressions".

Note in the example that the "colon" continuation applies to the virtual statement, i.e. to continue the current line end the line with a colon preceded by a space and indent the next line.

5.5.5 REQUIRE Statement

The REQUIRE statement is used to specify fields in the master file record and local fields which must be non-null[10] before the user can file the record to the disk. REQUIRE may apply to ER, DR, and link fields as well as fields in the active file.

REQUIRE is effective in three modes: Update, Insert, and Append. The presence of REQUIRE statements automatically invokes LFEXIT control in Update Mode even if the keyword LFEXIT or LFBACK is not on the screen header line (see Section 5.3.1.18 "LFEXIT or LFBACK: Update Mode Control"). Thus, in Update Mode, as well as Insert and Append Modes, REQUIRE ensures that the user can only write the record to disk via NEXT, and cannot leave the record until all required fields are entered.

In Update Mode, the LFBACK keyword (see Section 5.3.1.18 "LFEXIT or LFBACK: Update Mode Control") can be used to provide a way to leave the record without filing the current record. For example, if the user is unable to enter all of the required fields and therefore wants to backout of any changes to the record.

In Append Mode the user can always leave the record without writing anything to the file by pressing APND. However, to file the record in Append Mode the required fields must be non-null before the user can file the record with NEXT.

In Insert Mode, there is no way to leave the record except by pressing NEXT, which files the record.

Note that, as with LFEXIT control, REQUIRE is only activated after the user types into a non-key field on the screen. In Update Mode the user is not kept in a record with null required fields unless a non-key field in the record was typed into.[11] In Append or Insert Modes, or when the user enters any non-key field in Update Mode activating LFEXIT control, and then presses NEXT while any required field is null, TRANS displays the message "<field> IS REQUIRED". TRANS does not call the RMO, file the record, or go to the next record: the user must press the error key clearing the error condition to continue (when the error condition is cleared the cursor is placed at the REQUIRED field that caused the error condition.)

The REQUIRE statement is placed in the TRS field names section. If a REQUIRE statement refers to a field that is not in the active file, the field name must be defined in the field names section **before** that REQUIRE statement.

The REQUIRE statement has the following syntax:

```
REQUIRE field_1 field_2 ...
```

There may be any number of REQUIRE statements in a TRS, and each REQUIRE statement can specify up to 29 required fields.

REQUIRE statements may be used with single record screens only.

10. A non-null field is one that has a non-zero or non-blank value.
11. Fields identified in the ALLOW statement also do not activate LFEXIT control (see Section 5.5.14 "ALLOW statement").

The "<field> IS REQUIRED" messages occur in the order of the fields in the REQUIRE statements. This is not necessarily the same as the order of the fields in the field names section. For example, FLD1 might be the first editable non-key field on the screen and FLD6 might be the sixth. If the TRS contains the statement "REQUIRE FLD6 FLD1", and both of these fields are blank when the user presses NEXT, then TRANS displays the message "FLD6 IS REQUIRED". Then if the user enters a value in FLD6 but FLD1 is still blank, TRANS displays the message "FLD1 IS REQUIRED". Thus the screen designer can control the order in which the user is reminded to fill in required fields.

5.5.6 Check Statement

A Check statement is used to specify conditions under which the data being typed should be considered erroneous. There are two types of Check statements which may be used in a TRS for validating data entry. (1) The "C" Check statement which is evaluated each time data is entered into any field. And (2) the "CLF" Check statement which is only evaluated when the user presses NEXT to file a record. In Update Mode, the CLF Check statement is only evaluated when LFEXIT control is active (see Section 6.3.1.1 “Update Mode Under LFEXIT Control”).

C

C stands for check and is used to check for errors whenever data is entered into any field. C is followed by a conditional expression on the same line and an error message on the next line. For example:

```
C NET LT 0
NET NEGATIVE. EXEMPTIONS EXCEED
ASSESSMENTS.
```

This example considers it an entry error when entered data causes the value of NET to be less than zero. The error message "NET NEGATIVE. etc." is displayed when this occurs.

Any data entered at the terminal which causes a C conditional expression to be evaluated to "true" is considered an entry error. The entered data is not accepted, the error message is displayed at the bottom of the screen, and TRANS enters Error Mode which can only be cleared by the user pressing the ERR keystroke.

When a value is entered it is tested for all the intrinsic format checks, (i.e. does a decimal value contain alphabetic characters?). Then all the virtual fields and check expressions are evaluated. (A check expression may reference any virtual field name that precedes it in the field names section. Computed virtual fields need not be displayed on the screen; that is, virtual field names need not be included in the layout section.) If any of the check expressions are evaluated to be "true" then the entered data is rejected and all the virtual expressions are re-computed using the previous (correct) value. If, on the other hand, an entered value does not cause any check expression to evaluate to "true", then the value is accepted, and all the virtual fields are "refreshed" on the screen to reflect the newly entered value. (A more detailed discussion of the sequence of events that occurs when a value is entered can be found in Section 15.2.2 “Field by Field Processing: S$S = 'fieldname'”.)

Note that **every** C Check statement is evaluated each time data is entered into **any** field. Therefore C Check statements must take into account the possibility that the user has not yet gotten to the particular field on which the Check statement is operating. For example, if we are checking that amount paid (AMTPD) is not greater than amount due (AMTDUE), we should also check that these values are non-null, i.e. their data has been entered. For example:

```
C AMTPD GT AMTDUE AND AMTPD NE 0 :
AND AMTDUE NE 0 PAID EXCEEDS DUE
```

Note that the colon continuation is applicable to the expression portion of the Check statement.

CLF

CLF means "check at NEXT". CLF is usually used to validate data entry where checking relationships involves more than one field. CLF Check statements are not evaluated until the user presses NEXT to file the record.

CLF Check statements have the same syntax as C Check statements, except that the statement starts with the designator "CLF" instead of "C".

For example:

```
CLF APPR EQ '1' AND (DIST NE '1' AND
'2' AND '9')
DISTRIBUTION CODES ARE 1, 2 OR 9 FOR
APPROPRIATION CODE 1
```

In this example, a user might enter an appropriation code of '1' while the distribution code in the record is blank, without triggering the Check statement. The user continues to enter fields in the record, including the distribution code. The CLF Check statement is not evaluated until the user signals that all fields have been entered by pressing NEXT.

CLF Check statements can be used in screens with the APPEND, INSERT, LFEXIT, or LFBACK screen header keywords, and in screens which include REQUIRE statements (see Section 5.5.5 "REQUIRE Statement"). That is, the CLF Check statement operates in situations where the NEXT key is the only way to file the active record. If the screen does not have the APPEND, INSERT, LFEXIT, or LFBACK keywords but has only REQUIRE statements, then the first REQUIRE statement must precede the first CLF Check statement in the field names section of the TRS.

In Update Mode, CLF Check statements are ignored unless LFEXIT control is active. Note that if CLF Check statements are used in a screen in Append Mode, they will be ignored when the screen is in Update Mode unless LFEXIT, LFBACK, or REQUIRE is used to invoke LFEXIT control in Update Mode.

CLF Check statements may be used in conjunction with the table driven error message facility described in Section 5.5.6.1 "Table Driven Check Statement Error Messages".

C Check statements and CLF Check statements can be combined in a screen.

In applications with complex validation checking, the use of REQUIRE statements and CLF Check statements often allows screen designers to achieve a high degree of control with virtually no RMO programming. A very simple example is shown below. Note that LFEXIT control is implicit in this TRS example since the TRS contains a REQUIRE statement.

```
PO PO.MAS 1 NOMSG
*
LINK FUND.TAB
KC FUND
L DESC FDESC
END
...
```

```
*
E PO#
E FUND
E OBJECT
E APPR
E DISTRIB
E ITEM#
E QUANTITY
D AMT
E TOTAMT
D FDESC
...
 *
REQUIRE FUND OBJECT APPR ITEM# QUANTITY
TOTAMT
*
C QUANTITY LT 0
Quantity must be a non-negative value
*
* If the APPR (appropriation) code is
set to 1, then
* the DISTRIB (distribution) code must
be 1 or 2 or 9.
*
CLF (APPR EQ '1') AND (DISTRIB NE '1'
AND '2' AND '9')
Valid Distribution Codes are 1, 2 or 9
for Appropriation Code 1
*
CLF QUANTITY * AMT NE TOTAMT
Incorrect Order Total
*
C FUND NE 0 AND FDESC EQ ' '
Invalid fund code
PO PO.MAS 1 NOMSG
*
LINK FUND.TAB
KC FUND
L DESC FDESC
END


SCREEN
CE PURCHASE ORDER SCREEN
       PO#: PO------
        Fund: FUND-     FDESC---------
        Object: OBJ----   ODESC-------
Appropriation Code: AP-
Distribution Code: DIST-
Item: ITEM#-----  IDESC---------------
Quantity: -------QUAN
Amount: --------AMT
Total Amount: -----TOTAMT
END
```

This kind of error checking can also be achieved using an RMO behind the screen and the RJ$RJ local field (see Section 16.1.2 “Reject APPEND, INSERT, UPDATE, DELETE, or Transfer”). However, the CLF Check statement technique is preferable to the RJ$RJ technique both for the simplicity of CLF and because the CLF Check statement error message is situation specific.

5.5.6.1 Table Driven Check Statement Error Messages

The error message displayed when a Check statement condition is evaluated as true can be taken from an error message file, rather than including the error messages in the TRS instruction file. In order to do this, the first link paragraph in a screen must be to an error message file.

The error message table is an ADMINS data file keyed on an integer error message code number, with an alphanumeric message field. The message field may be of any length up to A80. (TRANS will display up to 72 characters on the error message line.) In addition, the error message table may contain any other fields required for documentation and control.

The first LINK paragraph in each screen links error messages from the table. The LINK paragraph must be in the following format:

```
LINK ERROR-MESSAGE-FILE-NAME
KC E$RR
L  E$RRMSG
END
```

E$RR and E$RRMSG are special reserved field names. E$RR must be an integer DR field in the TRS field names section. If the field containing the error message has a name other than E$RRMSG in the link file then it must be renamed to E$RRSMG in the link paragraph. For example if the message field is named MSG in the link file ERRMSG.TAB, the following link paragraph syntax should be used:

```
LINK ERRMSG.TAB
KC E$RR
L MSG E$RRMSG
END
```

There are two different ways to use Check statements to activate the link to the error message table. The method described below uses Check statements in the TRS to set the error code. Section 16.19 “Using the RMO with Table Driven Error Messages” describes the use of an RMO behind a screen to set the error code.

Check statements used for triggering table driven error messages are similar to other Check statements, and have the following format.

```
C CONDITIONAL EXPRESSION
ERR=n

CLF CONDITIONAL EXPRESSION
ERR=n
```

The Check statement uses the same syntax as described in Section 5.5.6 “Check Statement”, i.e. it begins with the letter "C" (check) or the letters "CLF" (check at NEXT) followed by a conditional expression. The second line of the Check statement, however, must begin with the 4 characters "ERR=" followed immediately by an error code value, with no embedded spaces. The error code value is a one to five digit number in the range of 1 to 32767, and should not contain a comma, or any other text. For example,

```
C EMPL# NE 0 AND LASTNAME EQ ' '
ERR=1201
```

If the condition is evaluated as true (e.g. the user enters an employee number and a last name is not found), TRANS uses the error code number as the link key to the error message file. If a record with that key value (e.g. 1201) is found in the link file, the field E$RRMSG is displayed at the bottom of the screen where error messages typically appear. If no such record exists in the error message table with that error code, or if E$RRMSG is blank in that record, then the message "ERR=n" is displayed.

This kind of table driven Check statement, as well as standard Check statements (both C and CLF), may be mixed within a screen.

5.5.7 Message Fields

M

M stands for message. M fields are used to display text messages on the screen when some condition exists in the active record. The field name specified after M is not from the record. Rather it is the name of a field which will appear in the screen layout section described below. For example:

```
M CRSTAT BALANCE - PRICE LT 0
CREDIT OVERDRAWN
```

This example says that when the credit balance value minus the price value is less than zero, display the message "credit overdrawn" in the display field called CRSTAT ("credit status") that will appear in the screen layout.

The general syntax for the M code is "M field-name conditional expression" on one line and the message of the text on the next line. A particular message field, e.g. CRSTAT, can be used in several Message statements, each with different messages for different conditions. That is, the Message statement is provided to allow the user to place one of several data dependent messages anywhere on the display screen.

The colon continuation is also applicable to the expression portion of the message statement.

5.5.8 Internal Fields

An internal field is a field maintained by TRANS to provide the active screen with information about current activity. Internal fields should be placed in the field names section as "DR" (display only "local" fields) if they are to appear on the screen. Internal fields are usable even if there is no record maintenance procedure behind the screen. However they are also accessible, via a local field definition, to the RMO.

The following internal fields are available in TRANS.

5.5.8.1 TODAY: Current Date

The field TODAY contains the current date[12]. (See Section 2.4.2 “Field Data Types” for a discussion of ADMINS date formats.) TODAY is included in the field name section as follows:

```
DR TODAY/DA

or

DR TODAY/DT
```

(If TODAY is used as a "local" field in the RMO running with the screen (as described in Section 15.1.3 “Local Fields in the RMO”) it **must** be included in the field declaration section of the screen description.)

5.5.8.2 NOW: Current Time

The field NOW is set with the current system time whenever a value is entered into any field on the screen. NOW is displayed in military time, i.e. HH:MM:SS if it is declared as field type A8 and HH:MM:SS.TT (TT is hundredths of a second) if it is declared as field type TM. NOW is included in the field name section as follows:

```
DR NOW/A8

or

DR NOW/TM
```

(If NOW is used as a "local" field in the RMO running with the screen (as described in Section 15.1.3 “Local Fields in the RMO”) it **must** be included in the field declaration section of the screen description.)

5.5.8.3 Terminal Number

The field T$T is loaded with the contents of the logical name ADM$TERM (see Appendix C.1.1 “Differences in Print File and Temporary File Naming”) which is usually the last two or three digits of the terminal number. For example, if the value of the logical name ADM$TERM contained the string "B2" for a user at a particular terminal, then T$T would contain "B2". T$T is included in the field name section as follows:

```
DR T$T/A4
```

(T$T is shown as an A4 type field. It may be also be an A2 or I type field.)

---

12. You can use the logical name ADM$TEST_TODAY to set “test values” for the TODAY and NOW fields. Set ADM$TEST_TODAY to a date in the format YYYY-MM-DD (this format is always used when assigning this logical regardless of the ADM$DATE setting). ADMINS will convert the specified value to the current date format (using ADM$DATE if assigned), E.g.
    AdmLcr ADM_TEST_TODAY 2009-12-24

    will set the internal field TODAY to 12/24/2009 instead of the current date, and set the internal field NOW to 23:23:23 instead of the current time of day (NOW is always automatically set to this value when this logical is assigned).

    ADM$TEST_TODAY is supported in all commands that support TODAY and NOW

5.5.8.4 D$IR: Default Directory

The D$IR field is an A24 field which is set to the user's default directory. For example, if the user's current default directory is [ACCTG], then D$IR would be set to "[ACCTG]". D$IR is included in the field names section as follows:

```
DR D$IR/A24
```

The D$IR subroutine returns both the device and directory specification as follows:

```
DISK: [DIR]
```

5.5.8.5 G$RP: UIC Group Number

The field G$RP[13] is an integer field containing the group number of the UIC under which the user is currently operating. For example, if the user's UIC is [65,30] then G$RP would be set to 65. G$RP is included in the field names section as follows:

```
DR G$RP/I
```

5.5.8.6 U$SER: UIC User Number

The field U$SER is an integer field containing the user number of the UIC under which the user is currently operating. For example, if the user's UIC is [65,30] then U$SER would be set to 30. U$SER is included in the field name section as follows:

```
DR U$SER/I
```

5.5.8.7 ADM$SCRNAM

If the reserved field ADM$SCRNAM/An is present, TRANS places the current screen name in it, and also assigns the current screen name to the process logical name ADM$SCRNAM. For example, if the current screen name is VEND_ENTRY then ADM$SCRNAM would be loaded with the value "VEND_ENTRY", and the string "VEND_ENTRY" would be assigned to the logical name ADM$SCRNAM in the process logical name table. ADM$SCRNAM is included in the field name section as follows:

```
DR ADM$SCRNAM/A16
```

5.5.8.8 ADM$TRONAM

The reserved field ADM$TRONAM/An works the same way as ADM$SCRNAM (see Section 5.5.8.7 “ADM$SCRNAM”), but for the TRO name. TRANS puts the current TRO name into ADM$TRONAM/An, and assigns the TRO name to the process logical name ADM$TRONAM. ADM$TRONAM is included in the field name section as follows:

```
DR ADM$TRONAM/A16
```

5.5.8.9 ADM$IDXNAME

If the reserved field ADM$IDXNAME/A30 is present in the TRANS virtual record it will be loaded with the Index name for the active index (and will be blank if the main index is in use). ADM$IDXNAME is included in the field name section as follows:

```
DR ADM$IDXNAME/A30
```

---

13. G$RP is a global field (see Section 5.5.9 “Global Fields”).

5.5.8.10 ADM$WINTITLE: Set Window Title

Including the special field **adm$wintitle/An** in the field declaraion section provides a way to set the title for the screen in the window "banner" (the default title that appears in the banner is "ADMINS WIN32 TRANS:" followed by the screen name).

This field can also be set or modified from the rmo.

An example:

```
n n.mas 1
e n
e fld
e d2
v adm$wintitle/a40 'This will be the title'
screen
      n       fld          d2
------n       fld-  -------d2
end
```

5.5.8.11 L%FIELD: Get Label for field from data dictionary

If a field name starts with "**L**%" and the rest of the name (e.g. ACCNT in L%ACCNT) exists as a field name in the screen's virtual record, and that field references an element in the data dictionary, the L%FIELD will be loaded with the value of the Line Label attribute for FIELD, and the field will be treated as a "Color Label" field (i.e. rendered with the "Label" color scheme as described in Section 5.5.16 "COLOR Statement").

For example:

```
ER ACCT/@ACCOUNT
DR L%ACCT/A20
.
.
.
SCREEN
L%ACCT--------   ACCT---------
```

5.5.8.12 X$MSG: Set Message in Status Bar

The reserved field name X$MSG/An is used to display an extended message in the first (left) part of the status bar of TRANS. The content of X$MSG will be appended to the regular content of the first part of the status bar, separated by " -- ".

5.5.8.13 ADM$CHKLCK: Check record locked status

If the special field ADM$CHKLCK is present in the TRO, and the screen has an RMO associated with it, then ADM$CHKLCK indicates if the current record is locked by another user. This check can be used when the main file is being accessed "read-only" or "multi-user[14]" (-R, -RX or -M).

If ADM$CHLCK is specified as an integer, the field is set to 1 if a record is locked and zero otherwise.

---

14. If the current record is being accessed multi-user (-M) in a single-record screen using this check may not make sense. If the record you are trying to access is locked you would be prompted "wait or ignore" before TRANS can evaluate ADM$CHKLCK. In multi-record screens however, ADM$CHKLCK can be used to display the lock status of all the displayed records, not just the current record.

If ADM$CHLCK is specified as an alpha field, the field is loaded with the nodename and username (e.g. \\nike\bobama) of the user who is locking the record, and is set to ‘ ‘ (blank) if the record is not locked. It is the responsibility of the developer to specify the size of the field sufficient to accomodate the largest string that could be loaded.

ADM$CHKLCK can be used, for example, in read-only screens to indicate to the user which records are locked by other users, so they can see which records are available for them to update or otherwise process.

5.5.9 Global Fields

Global fields allow the screen developer to hold information in TRANS as the user branches from screen to screen.

Global fields can be thought of as the "DEF" of a 1,024 16 bit word record that is constantly kept in memory. This record is only erased when the user exits TRANS completely. (Even the EXIT keystroke, see Section 6.8 “Control Functions”, does not erase the global record.)

TRANS treats field names that start with "G$" as global fields. TRANS maps these fields onto the global array in the order in which these fields appear in the screen description. Each global field occupies the number of words in the global record required to accommodate the ADMINS data type of the field. Every screen in an application which uses these global fields should contain the global field names in the **same order.**[15] The following example describes several global fields in a screen's field declaration section:

```
DR G$OPER/X9999          "word 1 of the global area"
DR G$BRANCH/A24          "words 2 thru 13"
DR G$BCH/I               "word 14"
DR G$TOTAL/D2            "words 15 thru 17"
```

It is important to understand how values in global fields are retained as TRANS branches from screen to screen. The contents of the global record is never altered in any way by TRANS itself. Only manual entry into a global field or an RMO behind the screen can actually change values in the global record.

As each screen is activated, TRANS sets a **pointer** to the first word of the global record. Then TRANS examines the field names in the screen **in the order they appear.** If a field name starts with "G$" TRANS sets that field name to point to the current word (as indicated by the pointer) of the global record, and increments the pointer to the global record by the number of words required to store that field, i.e. 1 word for an integer, or 10 words for an A20 field.

There may be occasions when part of the global area is not used by a particular screen description. There is a special notation used to bypass part of the global area, G$+nnn/I, where nnn is the number of words to bypass in the mapping process. For example, the first 200 words of the global area might be reserved for a special use and the application screen is to start using word 201 as follows:

```
DR G$+200/I              "skip the first 200 words"
DR G$OPER/X9999          "word 201 of the global area"
DR G$BRANCH/A24          "words 202 thru 213"
DR G$BCH/I               "word 214"
DR G$TOTAL/D2            "words 215 thru 217"
```

Therefore, if two screens contain the same global fields in the same order, these global fields will point to the same part of the global record as TRANS branches to each screen.

Since global fields are most commonly used with an RMO behind the screen, an example using global fields is in Section 15.4.3 “Example Using Global Fields”.

15. To insure that all screens in the application that utilize the global record use the same global fields in the same order, use a STRUCTURE paragraph (see Section 5.5.9.1 “STRUCTURE: Lay out global fields section”). For example, global field names used to rename a linked field in a LINK paragraph (see Section 5.4.1 “LINK Paragraph”) would be the first fields in the global record because they are encountered first (LINK paragraphs come before the field declaration section in the TRS). The sometimes tricky task of managing the order of global fields from screen to screen is completely eliminated by the use of a STRUCTURE paragraph.

5.5.9.1 STRUCTURE: Lay out global fields section

The STRUCTURE paragraph is used to impose a specific structure on the global record. As is explained in the previous section, global fields are mapped to the global record in the order that they are encountered in the screen description. The STRUCTURE paragraph is simply a means of making global fields known to TRANS in a specific order that is independent of how, in what order, or even if a field actually is used in a screen or its associated RMO. STRUCTURE paragraphs also allow the global record to be explicitly specified without having to declare each global field in the field declaration section, saving resources against TRANS' virtual record limits for maximum number of fields and maximum record size. (Global fields that are actually referenced in the current screen or RMO must still be declared as a DR or ER field for that screen, or declared as a local field in the RMO.)

Including an identical STRUCTURE paragraph[16] in the descriptions of all screens that use the global record **insures that the same global fields will be mapped in the same way** throughout the application.

The STRUCTURE paragraph must appear immediately after the header line in each field description.[17]

The syntax of the STRUCTURE paragraph is as follows:

```
STRUCTURE
G$fld1name/type
G$fld2name/type
G$fld3name/type
etc.
END
```

16. We recommend using "@@" indirect referencing to include the same STRUCTURE paragraph in each screen of an application. See Section 1.3.3 “Indirect References”.
17. If a VIDEO statement is present, the STRUCTURE paragraph follows the VIDEO statement.

5.5.10 BOX statement

The BOX statement (refer to Section 6.17.13.3 "BOX properties" for more information) provides an easy way to place graphics boxes and horizontal or vertical lines on the TRANS screen. The syntax is:

```
BOX  TOP_LINE  LEFT_COL  HEIGHT  WIDTH  [VIDEO_CODE]
```

All of the items in the BOX syntax are required, except VIDEO_CODE.

TOP_LINE

LEFT_COL

The first two items in the BOX statement provide the line/column coordinates of the upper left corner of the box TRANS is to display.

HEIGHT

WIDTH

The third and fourth items provide the vertical and horizontal dimensions of the box. HEIGHT is specified as the number of lines. WIDTH as the number of columns.

VIDEO_CODE

The optional BOX video code enables you specify video attributes for the box. BOX video codes are taken from the sum of four settings:[a]

1	bold
2	underscore
4	blink
8	reverse video

For example, video code 9 produces both bold and reverse video.

Reverse video boxes without border lines can be created by making the video code negative (for example, the video code -8 creates a rectangle entirely of reverse video, with no graphics line around it).

If high intensity (bold) highlighting is used, the borders of the box are displayed in high intensity. Whatever is inside the box is displayed in normal video.

If reverse video is used for the box, the entire box appears in reverse video. Literals and fields inside the box appear in normal video.

a. These video codes are also used with the H$CODE highlighting facility (see Section 16.5 "Highlighting Fields")

BOX can be used to produce vertical and horizontal graphics lines. For a vertical line, specify a width of one column; for a horizontal line, specify a height of one line.

TRANS displays boxes and lines in the order of the BOX statements, before it displays literals or data. Take note of the following when designing screens that use BOX statements:

1. A box will overlay an earlier box or line if it should overlap.
2. Each successive box blanks out whatever was previously inside it.
3. When boxes or lines intersect, TRANS joins them with corner, tee, or cross characters.
4. Literals and fields on the screen should not overlap with the boundaries of a box, or with a line, because the characters will overwrite the image.

Up to 250 BOX statements are allowed per screen.

5.5.10.1 Drawing BOXes in the screen layout

AdmScreen also allows the developer to "draw" boxes and lines in the SCREEN layout section, instead of specifying them using coordinates and dimensions.

To draw boxes in the screen layout, you use three reserved characters. These characters can be anything you want (explained below), but by default AdmScreen recognizes the following characters for drawing boxes: '=' for horizontal lines, '!' for vertical, and '+' for any type of intersection. For example:

```
+===============+
!               !
! Hi There      !
!               !
+===============+
```

The intersection character is used for **any** type of intersection between two lines, whether it is a corner, a T junction, or a cross junction.

There is also a default video attribute setting for BOXes drawn in the screen layout. The built-in default video attribute is normal video.

In the layout, you can put special BOX characters right next to literals and field descriptors. The following layout would work:

```
+===============+
!               !
!FIELD-  LITERAL!
!               !
+===============+
```

To activate this feature using the built in defaults, you must put the statement BOX DEFAULT above the SCREEN statement. This tells AdmScreen to give a special interpretation to the three reserved characters when it parses the layout. If BOX DEFAULT (or another special BOX statement as described below) is not present, then all characters in the layout are interpreted in the normal way.

Notes on “drawing” boxes:

1. You can draw boxes in the layout, and also use standard BOX statements with coordinates, in the same screen. These two ways of boxing are compatible and have no effect on each other.
2. All boxes drawn on a given screen layout will have the same video attribute: if you need boxes with different video attributes on the same screen, you must use standard BOX statements, or a mixture of standard BOX statements and drawn boxes.
3. AdmScreen transforms the box layout into a series of coordinates and dimensions, and the result is exactly the same as a set of standard BOX statements. Boxes drawn on the screen count against the limit of 250 BOXes per screen.

To override the built-in defaults globally, you can make an assignment to the logical name ADM$SCR_VIDEO (which is used for a similar purpose by the VIDEO feature).[18] ADM$SCR_VIDEO takes effect only at screen compile time (not at run time in TRANS). ADM$SCR_VIDEO options for BOX defaults are:

```
HORIZONTAL=h
VERTICAL=v
INTERSECT=i
VIDEO=<video attribute keyword(s)>
```

These options can be combined in ADM$SCR_VIDEO with the options for the VIDEO feature. The HORIZONTAL, VERTICAL, INTERSECT, and VIDEO keywords can all be abbreviated to two characters and they are case blind. Any or all of these four options can be used and they can be in any order. There cannot be any embedded blanks in a keyword string (i.e., no blanks around the '=' signs). The three BOX characters represented above as 'h', 'v', and 'i' can be any printable characters, except you cannot use '-' or '*'; and the three characters must be different after all defaulting occurs. The video attribute keywords are the same as for the VIDEO feature: BOLD, UNDERLINE or UL, REVERSE, BLINK, NORMAL, and FULL, and they can be combined with '+' as for VIDEO (no blanks around the '+'). Here, FULL means don't draw a border around the box (same as using a negative video code in a standard BOX statement).

You can also change the BOX defaults on a per-screen basis. To do this, instead of the BOX DEFAULT statement, put the new defaults in a special BOX statement:

```
BOX HORIZONTAL=h VERTICAL=v INTERSECT=i VIDEO=<keywords>
```

Syntax rules for this BOX statement are the same as for ADM$SCR_VIDEO.

Defaulting work as follows.

1. Built-in defaults are assumed.
2. Built-in defaults are modified by the ADM$SCR_VIDEO assignment.
3. The result of 1 and 2 is modified by special BOX statements for a specific screen.

5.5.11 CAPS statement: Convert entry to all uppercase

The CAPS statement causes specified editable fields to be converted to uppercase as each character is entered.

The CAPS syntax is:

```
CAPS FLD1 FLD2...
```

where FLDn are editable fields which appear **above** the CAPS statement in the field names section of the screen. There can be any number of CAPS statements.

18. See Section 5.10 "Video Highlighting Facilities" for a discussion of the VIDEO highlighting facilities, and the logical name ADM$SCR_VIDEO)

5.5.12 CAP1 statement: Capitalize each word in entry

The CAP1 statement causes specified editable fields to be converted so that beginning letter of each word in the entry is converted to uppercase upon completion of the entry.

Except that CAP1 conversion does not occur until the entry into the field is complete, CAP1 syntax and use is identical to CAPS, described in the previous section:

```
CAP1 FLD1 FLD2...
```

5.5.13 NOECHO

The keyword NOECHO in a TRS allows you to list one or more fields in which the content displays as a string of asterisks. The syntax is:

```
NOECHO FIELD1 [ FIELD2 …]
```

As you type a new value into the field each character echos as an asterisk (the actual characters you type are stored in the field). Useful when entering secure items such as passwords.

5.5.14 ALLOW statement

The ALLOW statement identifies fields where typing is to be allowed in those circumstances when field entry would normally be restricted or prevented, as follows:

1. When the value "Y" has been assigned to the logical name ADM$READONLY (see Section 6.4 “Entering or Changing Fields”).

   With ADM$READONLY set all files are opened "read-only". Nothing can be changed in any of the files. The cursor normally goes only to KEY fields. In this case, the cursor will also go to ALLOW fields to allow entry (nothing is actually written to disk).
2. When LFEXIT (see Section 5.3.1.18 “LFEXIT or LFBACK: Update Mode Control”) or REQUIRE (see Section 5.5.5 “REQUIRE Statement”) is used in a screen, ALLOW fields may be changed without activating LFEXIT control.

In general, ALLOW fields should be used only to cause TRANS to perform some action, such as an automatic branch. When the ALLOW feature is used to override ADM$READONLY or LFEXIT processing, **values typed into ALLOW fields are not written to disk.**[19]

The ALLOW statement syntax is:

```
ALLOW FIELD1 FIELD2....
```

where the FIELDn is the name of a field in the active file, or the name of a field that has been declared for the screen prior to the ALLOW statement.

19. A value typed into an ALLOW field could be written to disk if normal LFEXIT processing is used to subsequently write the record.

5.5.15 LABEL Statement

The LABEL statement allows you to specify the font and the color for labels (literal text) in the screen. The syntax is as follows:

```
LABEL [y,x] 'Label text' FONT ##  TEXTCOLOR color
```

where:

LABEL	can be abbreviated to LA.
[y, x]	the line and column number where the label text starts
'Label text'	any text (starts at line y, column x)
FONT ##	Optional. A font specification where ## is a number between 20 and 29 (see Section 6.17.13.2 "WINDOW keywords")
TEXTCOLOR color	Optional. A specification of the text (foreground) color of the text (e.g. **TEXTCOLOR red**).

5.5.16 COLOR Statement

To determine which fields takes which color attributes, SCREEN will accept one or more COLOR statements with the following syntax:

```
COLOR xx FIELDNAME1 FIELDNAME2 …
```

where 'xx' is one of the keywords 'key', 'edit', 'display', 'label', 'msg', 'message', or a number between 20 and 99.

Keywords refer to the color scheme currently in effect for a particular field category, for example:

```
COLOR label labaddr1 labaddr2
```

specifies that fields labaddr1 and labddr2 should be rendered with the color scheme for label fields (either the default scheme or the one specified in the TRANS Environment File).

Similarly, numbers correspond to a color-scheme number specified in the TRANS Environment File, for example:

```
COLOR 37 labaddr1 labaddr2
```

specifies that fields labaddr1 and labaddr2 should be rendered according to color scheme 37, as specified by "field.color_NN..." entries in the TRANS_ENV file (see Section 6.17.13.4 "FIELD keywords").

5.5.17 BORDER statement

The BORDER statement allows the developer to assign alternative field window border styles to specifically named fields in the screen.

```
BORDER   NONE|RAISED|SUNKEN|ETCHED|BUMPED  fieldname1, filedname2…
```

For example:

```
BORDER NONE NAME 1ADDR 2ADDR CITY STATE ZIP
```

5.5.17.1 DISPLAY_BORDER NONE statement

Like the BORDER statement described in the previous section, the DISPLAY_BORDER NONE statement allows the developer to assign a specific border style to a specified list of fields, but DISPLAY_BORDER NONE is for a special case.

DISPLAY_BORDER NONE allows the developer to declare that when the named fields are switched from editable to display-only via the EDFLDS subroutine (see Appendix H.13.7 "EDFLDS - Modify List of Editable Fields in TRANS"), the fields will display without a border.

For example:

```
DISPLAY_BORDER NONE NAME 1ADDR 2ADDR CITY STATE ZIP
```

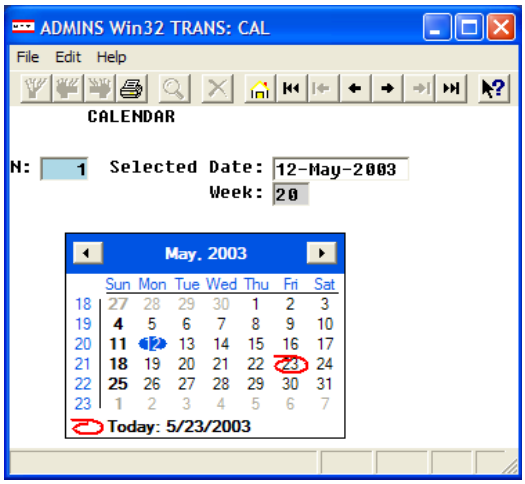
5.5.18 TOOLBAR Statement

Use the following TOOLBAR statements to control display and appearance of the TRANS toolbar
.:

TOOLBAR OFF	Turn TOOLBAR BUTTONS off
TOOLBAR STRINGS	Turn text on in toolbar buttons
TOOLBAR NOSTRINGS	Turn text off in toolbar buttons (bitmaps only)
TOOLBAR BUTTONS btn#, btn#	Define toolbar buttons for this screen. Btn# is a valid button number (see Section 6.17.13.10 "Toolbar"), or a negative number for separator(s).

5.5.19 Calendar Keyword

USe CALENDAR to display a calendar control in the screen:



The syntax is:

```
CALENDAR Y X DATEFIELD [ WEEKNO[=IFIELD] ]
```

where:

Y and X	Specifies the line and column number for the upper left corner of the control. TRANS automatically calculates how to properly display the calendar control (approximately 9 lines and 26 columns) to avoid having it cover (or be covered by) other objects on the screen]
DATEFIELD	A field of type DA or DT that receives the date when one is selected by clicking on it in the control.
WEEKNO	(Optional) If present, this keyword displays week number in the control. If WEEKNO is immediately followed by =FIELDNAME, where FIELDNAME is of type I, the week number is returned in this field when a date is selected.

The CALENDAR is a standard Microsoft Windows control, and adjusts its appearance according to the current user's Windows "locale".

To move to a different month, you may either click on the arrow buttons to move one month at a time, or you may click on the month name to get a pop-up menu listing all the months in a year.

To move to a different year, click on the year next to the month name, and an up-down control appears next to the year.

Sundays are displayed in **bold**. To have holidays (moveable and fixed) also display in bold, make a list of holiday dates on the form 'yyyy-mm-dd' in a text editable file, and assign its pathname to the logical name ADM_CAL_HOLIDAYS. E.g.:

```
0000-01-01
0000-07-04
0000-12-25
2008-05-26
2008-09-01
2008-10-13
2008-11-27
```

To specify fixed holidays use year = 0000. In the example above January 1, July 4 and December 25 will display in bold in every year. For 2008, May 26 (Memorial day), September 1 (Labor Day), October 13 (Columbus Day), and November 27 (Thanksgiving Day) will display in bold.

A maximum of 200 holidays may be present in the file. The file may contain any number of comment lines starting with '!' or '*' in column 1.

The date may be followed by white space and the '#' character followed by a number, e.g.

```
0000-01-01  #1  ! New Years Day
0000-07-04  #1  ! Constitution Day
0000-12-25  #2  ! Christmas day
2008-05-26  #3  ! Memorial day
2008-09-01  #3  ! Labor Day
2008-10-13  #3  ! Columbus Day
2008-11-27  #3  ! Thanksgiving
```

You may use this number to classify your holidays (or other days) in any way you want, e.g. 1 = national holyday, 2 = religious Holiday, etc. Anything that follows the date and a possible #number on the line is ignored.

If there is an RMO behind the screen, the RMO is called with M$M set to 'CA' whenever a date is selected in the calendar control.

If the RMO contains the field ADM$CALINFO/I or the array ADM$CALINFO/i(n) the day of the week (Monday=1, Sunday=7) of the selected date is returned in ADM$CALINFO or ADM$CALINFO(1). If the dimension of ADM$CALINFO is greater than 1 ADM$CALINFO(2) is set to the "#" number following the date in the ADM_CAL_HOLIDAYS file, or to 0 (zero) if no such number exists.

5.5.20 TIMEOUT statement

TRANS' time-out facility allows the application developer to set a maximum amount of time that a particular screen session[20] can remain inactive, i.e. with TRANS waiting for the user to type another character.

This feature is implemented via an integer field called "G$TMO[21]". G$TMO is set[22] to the number of seconds TRANS should wait for a character to be typed before timing out.

When TRANS "times out" the screen, TRANS will generate the keystrokes specified in the TIMEOUT statement, if one exists. Otherwise, TRANS issues a call to the RMO with S$S set to 'G$TMO'. The TIMEOUT statement or the RMO can then take an appropriate action, e.g., execute a branch or exit (see the example below).

The TIMEOUT statement consists of the keyword TIMEOUT followed by specification of the keystroke macro that is to be execute at time-out. Macro syntax is identical to that used in the *define* statement in the TRANS environment file, described in Section 6.17.2 "Define Macro Function".

If G$TMO is reset to zero then the time-out facility becomes inactive.

20. Timeout can be implemented globally, via the ***global.timeout*** setting in the TRANS Environment File (see Section 6.17.7 "Global Timeout").
21. Because "G$TMO" starts with "G$" it will map into TRANS' global fields area (see Section 5.5.9 "Global Fields") if it appears in the TRS as a DR field. In this way it can be set in one screen and then remain active in subsequent screens.
22. G$TMO can be set via the RMO or be declared as a Virtual Field (see Section 5.5.4 "Virtual Fields").

5.5.20.1 Time-out Examples

The following examples specify that if TRANS has waited for 5 minutes (300 seconds) for the user to type another keystroke, then TRANS will exit..

Implementing time-out via the RMO: TRANS is to exit via B$B= ‘CB’ (see Section 16.2.4 “Automatic Exit From TRANS: B$B = 'CB'”).

```
...
S$S/A6
B$B/A2
G$TMO/I 300 ! if TRANS waits 300 sec ‘G$TMO’ RMO call is issued
PROGRAM
IF S$S EQ 'G$TMO' THEN B$B = 'CB' ; GOTO OUT END
...
OUT: STOP
```

Implementing time-out entirely within the TRS, using the TIMEOUT statement: TRANS exits via keystroke macro that consists of the TRANS *EXIT* key and the *RETURN* key.

```
...
V G$TMO/I 300     ! if TRANS waits 300 sec TIMEOUT macro executes
TIMEOUT %exit CR  ! exit followed by return to take user to DCL
prompt
...
```

5.5.21 Button Objects in TRANS

AdmTrans supports PUSHBUTTON, CHECKBUTTON, and LABELBUTTON objects. PUSHBUTTONs and LABELBUTTONS can be specified completely in the TRS, or may be created and/or modifed by the BUTTON[23] subroutine, as described in Section H.13.3 “Button - Creating and Modifying Buttons in TRANS”

5.5.21.1 PushButton Object

PushButtons in TRANS place labeled "buttons" on the screen that may be mouse-clicked on by the user to initiate a specified TRANS keystroke macro. PushButtons may be located at specific screen coordinates or they may be placed relative to another button or field. A PushButton initiates the same series of keystrokes every time it is clicked on.

The PUSHBUTTON keyword may be written as BP.

The syntax for a PushButton is:

```
PUSHBUTTON Button_name [ %videocodes ] ['Label text' ]

NOTE: all following must be on new lines, indented
-------------------------------------------------
[ ROW=row# ]              ! Must be present if no
                          ! ATTACH or Layout placement
[ COLUMN=column# ]        ! Must be present if no
                          ! ATTACH or Layout placement
[ ATTACH_TOP=Button or Field ]
[ ATTACH_LEFT=Button or Field ]
[ ATTACH_RIGHT=Button or Field ]
[ ATTACH_BOTTOM=Button or Field ]


[ LABEL=Label_string ]    ! Literal label to be placed on
                          ! the Button. If not present, the
```

23.CHECKBUTTONS cannot be created or modified via the BUTTON subroutine

```
                          ! Button name will be used
[ HOVERTEXT=Hover_Text_string ]
                         ! Hover text for button
                        ! (repeat for multiple lines of hovertext)
[ ACTION=Actions... ]    ! Keyboard actions to perform
                         ! when pressed
[ WIDTH=n ]
[ HEIGHT=n ]
[ BITMAP=pathname ]
[ BITMAP_SIZE=size ]
[ ICON=number|pathname ]
[ ALIGN=left | right]         ! Center is default for text
[ BITMAP_ALIGN=center | right ! Left is default for image
[ FONT=FONT# ]
[ COLOR=foreground[,background ]
[ GRAYED_COLOR=foreground[,background ]
```

ROW =

COLUMN = The row and column number for the PushButton's upper left corner. ROW and COLUMN cannot be used if any ATTACH statement is used. Instead of using ROW/COLUMN, buttons may be placed in the layout section of the screen like a field, using the button name as a "field" name, e.g.

```
MYBUTTON--
```

ATTACH_TOP= Used to attach the top of the PushButton to the bottom of another Button or a field.

ATTACH_LEFT= Used to attach the left side of the PushButton to the right side of another Button or a field.

ATTACH_RIGHT= Used to attach the right side of the PushButton to the left side of another Button or a field.

ATTACH_BOTTOM= Used to attach the bottom of the PushButton to the top of another Button or a field.

LABEL= The character string that will be placed inside the PushButton. If it contains more than one word, it must be surrounded by single quotes, e.g.:

```
LABEL='PRESS ME'
```

Label may also be specified on the PUSHBUTTON (BP) line:

```
PUSHBUTTON NEXT %REVERSE+BOLD ‘Next PO‘
```

The width of the label determines the width of the button. If no LABEL statement is present, the PUSHBUTTON name will be used.

HOVERTEXT= “Hover text” that will “pop up” when the mouse cursor passes over this button. Use multiple HOVERTEXT statements for multiple lines of hover text.

WIDTH=n Width in characters. Used to create a button wider than would be created using the LABEL.

HEIGHT=n Height (in lines of screen display)

ACTION= One or more keystrokes to execute when released. Uses same syntax as when defining keyboard macros in TRANS$ENV. E.g. to execute branch 4 when a PushButton is pressed, use:

```
ACTION=%brnc 4
```

BITMAP	Pathname of bitmap to be painted on the face of the button.
BITMAP_SIZE	If the bitmap size is not 32x32 pixels, its size in pixels must be given using the BITMAP_SIZE keyword to be able to scale the bitmap correctly to the size of the button. If the horizontal and vertical size is the same, only one number is necessary.
ICON	If a number, "icon number" of known icon to be painted on the face of the button. If an alpha string, the pathname of the icon to be painted.
ALIGN	By default the text label is centered on the button. Use ALIGN=RIGHT or ALIGN=LEFT to have the label right or left justified on the button.
BITMAP_ALIGN	By default a bitmap or icon is aligned on the left of the button. Use BITMAP_ALIGN=CENTER or BITMAP_ALIGN=RIGHT to have the bitmap or icon centered or aligned on the right of the button.
FONT	Specify a user defined font number to use for the literal text in the button. **font=font#** where font# is a user defined font number in the range 20-99 as defined in the in-effect TRANS_ENV file.
COLOR	By default, TRANS creates a "windows" button. Use COLOR to have ADMINS create "custom" buttons with custom colors.
GRAYED_COLOR	When custom buttons are created, you can also specify how these buttons should be displayed when they are in a "grayed" (e.g. disabled) state.

5.5.21.2 Checkbutton Object.

ADMINS on Win32 accepts a new button type, CHECKBUTTON, which is a button with the ability to display a check mark, and a label. This type of button can only take two states, CHECKED or UNCHECKED.

A CHECKBUTTON must be tied to a field, which must be an A2 or I. If the data type of the field is A2, the default values for the two states are T for checked (true), and F for unchecked (false), and if the data type is I, 1 for checked and 0 for unchecked.

The field which the CHECKBUTTON is tied to must be declared as editable (E FIELDNAME), but it does not have to appear on the screen.

When the user clicks on the button, TRANS acts as if the user typed the new value in the attached field.

The general syntax is:

```
E MYFIELD
…
CHECKBUTTON BTNNAME 'Buttonlabel'! CHECKBUTTON or BC
    FIELD=MYFIELD
    [ CHECKED=checkedvalue ]
    [ UNCHECKED=uncheckedvalue]
```

CHECKED and UNCHECKED values only need to be present if you want other values than the default.

A CHECKBUTTON cannot have ACTION codes, but most other keywords you can use for PUSHBUTTONS can be used for CHECKBUTTONS.

5.5.21.3 LabelButton Object

LabelButtons are used to place literal text on the screen. The syntax for a LabelButton is:

```
LABELBUTTON Button_name [ 'Label text' ]
      [ ROW=row# ]          ! Must be present if no ATTACH
                            ! or Layout placement
      [ COLUMN=column# ]    ! Must be present if no ATTACH
                            ! or Layout placement
      [ ATTACH_TOP=Button or Container ]
      [ ATTACH_LEFT=Button or Container ]
      [ ATTACH_RIGHT=Button or Container ]
      [ ATTACH_BOTTOM=Button or Container ]
      [ LABEL='Character string' ]
```

The LABELBUTTON keyword may be written as BL.

5.5.21.4 Using Button Fields

Buttons fields have flexible syntax options that allow developers to code applications in whatever style best suits their needs.[24] For example, these two code fragments produce an identical set of buttons:

```
                          Fragment #1
!Next Button
BL LNXT  'Next PO         '!Label will be left
    ROW=7                  !of clickable button
    COLUMN=2               !All other buttons "hang" off LNXT
                           ! i.e. if it is moved others will keep
                           ! position relative to it.
BP NEXT  '  '      !Blank
   ATTACH_LEFT=LNXT        !attached right of label
   ACTION=%next            !does %next keystroke
!Prev Button
BL SEP1 ' '                ! Blank label button
   ATTACH_TOP=LNXT         ! used as separator
BL LPRV  'Previous PO     '! Label
  ATTACH_TOP=SEP1
BP PREV  '  '      !Blank
   ATTACH_LEFT=LPRV        !
   ACTION=%prev            !does %prev keystroke


                          Fragment #2

!Next Button
LABELBUTTON LNXT           !Label will be left
   ROW=7                   !of clickable button
   COLUMN=2                !All other buttons "hang" off LNXT
                           ! i.e. if it is moved others will keep
                           ! position relative to it.
   LABEL='Next PO         '
PUSHBUTTON NEXT            !Blank
   ATTACH_LEFT=LNXT        !attached right of label
   LABEL='  '
   ACTION=%next            !does %next keystroke
!Prev Button
LABELBUTTON SEP1           ! Blank label button
   ATTACH_TOP=LNXT         ! used as separator
   LABEL=' '
LABELBUTTON LPRV           ! Label
   ATTACH_TOP=SEP1
   LABEL='Previous PO     '
PUSHBUTTON PREV            !Blank
   ATTACH_LEFT=LPRV        !
   LABEL='  '
```

24. The action code for a button can be specified as M$M_nn. For more information, refer to Section 15.1.2.1 “M$M_nn: Action Code For Button”

```
    ACTION=%prev              !does %prev keystroke
```

The buttons produced by either of the above code fragments will look like this:

```
Next PO
Previous PO
```

5.5.22 %PUSHBUTTON: Display Alpha field as PUSHBUTTON

An alphanumeric field (An) can be displayed as a pushbutton. The field's value will be the button's label. The syntax is:

```
E FIELDNAME %PUSHBUTTON
    ACTION=action(s)
```

For example:

```
e invoice %pushbutton
    action=m$m_55
```

IN the above example, setting ACTION=M$M_55 specifies that the RMO will get a call with M$M set to '55' when the button is clicked. %PUSHBUTTON may be used in multi-record screens.

5.5.23 %CHECKBOX: DIsplay field as Checkbox

Use "%CHECKBOX" to display a checkbox for a field that can only take on two values (TRUE/FALSE, ON/OFF, checked/not checked). The syntax is:

```
E FIELD %CHECKBOX
```

When you place the field on the screen it will display as a checkbox. The box is unchecked if the field has a null value (zero, or blank for A fields), and checked if it has a non-null value.

5.5.24 %RADIOBUTTON: Display field as Radio Buttons

USe %RADIOBUTTON to display radio buttons for a field that will take on a limited number of different values. The syntax is:

```
E FIELD%RADIOBUTTON
    [ VALUE CODE [ LABEL ]]
    …
    [ HORIZONTAL ]
```

For example:

```
ER COLOR/I %RADIOBUTTON
    VALUE 0 &White
    VALUE 1 &Blue
    VALUE 2 &Red
    VALUE 3 Blac&k
```

By default, AdmTrans arranges the radio buttons vertically (one radio button with label per line). If the HORIZONTAL keyword is present the radio buttons are arranged horizontally with no labels (the developer must provide the appropriate labels above or below the radiobuttons).

Note that Windows "accelerator keys" can be specified for the radiobutton value choices by preceding the desired letter in the value choice's label with an ampersand "&". In the example above, the accelerator keys specified for radiobutton value choices "White", "Blue", "Red", and "Black" are "W"," B"," R" and "K"

If no VALUE clauses are present, a codelist tied to the field via the Data Dictionary is used.

5.5.25 %COMBOBOX: Display field as Combo Box

Use %COMBOBOX to display a field as a list of values from which the user can select. The value selected is what is stored in the field.

The general syntax is as follows:

```
E/D FIELD %COMBOBOX
    [ DROPDOWN  n ]
    [ VALUE Code Description ]
    [ VALUE Code Description ]
    ...
```

Where:

DROPDOWN n	Specifies how many items are displayed in the dropdown box when it opens (more items may be viewed by **ing.
VALUE Code Description	Specifies a number of **Code** values and their corresponding **Description** values that are used with the combo box field. The **Code** value must correspond to the data type of the field, as this is the actual value that is stored in the field when it is selected. The **Description** is any alphanumeric string used to describe the code. **NOTE:** If either the **Code** or the **Description** contains white space, it must be surrounded by quotes, e.g. 'This is the description'. If the field is tied to a codelist table, and no **VALUE** clauses are given for the **COMBOBOX** field, the codelist table is used to provide Code/Description entries (internal or external codelists allowed). **NOTE:** The Code is what is stored in the field when the item is selected, but only the Description is displayed on the screen. If you want to display the code as well, it must be repeated in the description.

5.5.26 SELECT Statement in TRANS

The SELECT statement provides a way for TRANS to operate on a "virtual" file of records that meet the selection criteria. The syntax is:

```
SELECT [NO]LINK [%EXECUTE XXPP] select statement
```

Where:

[NO]LINK	determines whether links are performed before the select statement is executed or not. One of these keywords are mandatory. If the select statement only references fields in the main file use the NOLINK keyword, as executing the links in the select phase adds a lot of overhead.
%EXECUTE XXPP	determines if the RMO should be called before the links are executed. This may be necessary if the RMO usually creates link keys in the pre-link RMO call. If present, the RMO will be called pre-link with M$M set to the first two characters of the word following the %EXECUTE keyword (i.e. "XX" in the example to the left), and the RMO will be called post-link with M$M set to the last two characters of the string following the %EXECUTE keyword(i.e. "PP" in the example to the left). If only two characters are present only the pre-link call will occur. If just a post-link RMO call is needed, put in ".." (two dots) for the pre-link M$M value, e.g. "%EXECUTE ..PP". It is the developers responsibility to ensure that these M$M settings are unique. Only calculation of fields that are necessary for the links (or the SELECT statement) should be done in these RMO calls.

When a selected record is present in the TRS, a snapshot of records that satisfied the selection criteria at the time of executing the select statement is presented to the user. The records on the screen remain selected until the user moves to another screen-full of records, even if somebody changed some value(s) in a record so that it no longer satisfies the selection.

If a SELECT statement is present, S$SEL has no effect.

The file should have a "dummy" or NULL record, as the first record in the file displays if no records are selected.

It is important to understand that no regular RMO calls are executed in the selection phase (with the possible exception of any RMO calls caused by the %EXECUTE part of the SELECT statement, and no V (virtual field) logic is executed. Any calculations or logic necessary for the selection criteria must therefore be contained within the SELECT statement itself, or possibly set during the %EXECUTE RMO call.

5.6 Screen Layout

The fourth component of the screen description is called the screen layout section. This section starts with the word SCREEN on a line. The word SCREEN can (optionally) be followed by four numbers that "position" the layout on the screen. The first two numbers are the line and column number for the upper leftmost corner of a rectangle where the screen layout should be displayed. The second two numbers are the number of lines and the number of columns of the display rectangle. If these four numbers are absent, they are assumed to be 1, 1, 24, 80. (They are assumed to be 1, 1,

24, 132 if the screen header line contains the keyword 132, and 1,1,LL,WWW if the screen header line contains the screen size (the LLxWWW keyword). The SCREEN line, showing the default position, is written as follows:

```
SCREEN 1 1 24 80
```

By default, the total usable space on the display screen consists of 24 lines of 80 characters each. The screen layout can be 132 characters wide if the screen header line contains the keyword 132 and the terminal supports 132 character lines.The lines that follow the SCREEN line contain the layout of what is to be displayed on the terminal when this particular screen description is activated. The conventions for the layout are very similar to those used to specify the layout for the DETAIL section of a report. Namely:

1. Each line corresponds to a line on the terminal.
2. Strings that contain neither leading nor trailing dashes are treated as literal text and displayed exactly as they are entered in the screen layout.
3. Strings with leading or trailing dashes are treated as (partial) names for fields from the field names immediately preceding the layout. Leading dashes indicate right justification of the data in the space occupied by the (partial) field name following the dashes. Trailing dashes indicate left justification.
4. Fields that are designated to be smaller than the full width for the type of the field (for example a decimal field designated to be only eight characters wide, or an A60 field designated to be only 25 characters wide) are truncated to the designated width (and thus might not show the entire contents of the field).

   When this happens AdmTrans automatically will display the entire contents of the field in a hover text window when the mouse cursor hovers over the field in the active record.
5. (During data input, leading blanks are usually squeezed out of alphanumeric fields. However if T is included in the string assigned to the logical name OPTION, then leading blanks are preserved in alphanumeric fields. See Appendix A: “Options”.)
6. If a field designator for a text field[25] has a number embedded in its string of dashes, the number is treated as the height of a multi-line block of text from the text field to be displayed starting at the position of the field designator. The height can be anywhere within the dashes (but it must be surrounded by dashes). For example, to display up to 6 lines of text stored in the TI field ADDRESS:

   ```
   ADDRESS-------------6---
   ```

   The height specified must be non-zero and positive. It is the responsibility of the developer to ensure that text blocks do not overlap other items in the screen display.
7. If a field designator for an editable alpha (An) field has the letter “S”, or a string containing “Snn” where nn is a number, embedded in its string of dashes, you will be able to enter more text than can be displayed in the field width designated by the string of dashes. For example:

   ```
   ADDRESS---S---
   ```

   When entering data 

   the data will scroll out of view to

25. See Section 5.16 “Text Fields”.

the left, 

allowing more characters to be entered. Precise placement syntax for this feature is:

```
ER NAME [5,16,20,S]
or
ER NAME [5,16,20,S30]
```

8. The letters BL in the first 2 character positions in a line indicates the line is to be displayed as all blanks.
9. The letters CE in the first 2 character positions in a line followed by a blank indicate that the remaining information in the line is to be centered within the dimensions of the current screen. A line with CE may contain both data fields and/or literal text. Centering of variable length data fields by AdmScreen is based on reserved length, not on the length of the actual data.
10. The letters DW in the first 2 character positions in a line followed by **two** blanks indicate that the information on that line is to be displayed using double width characters. Data and literal text may be displayed using DW. The double width character capability is a feature of the DEC VT series terminals.
11. The letters DH in the first 2 character positions in a line followed by **two** blanks indicate that the remaining information in the line is to be displayed using double height double width characters. Literal text may be displayed using DH. The double height, double width character capability is a feature of the DEC VT series terminals. When DH is used, two lines of the layout must be identical except for the DH; the first line for the upper half of the character and the second line for the lower half of the character. For example:

```
SCREEN
DH      Personnel Information Screen
        Personnel Information Screen
...
```

5.6.1 Precise Placement of Fields

Screen layout information for fields can also be supplied on the field declaration line as follows:

```
[LINE,COLUMN,SIZE[,JUSTIFICATION]]
where:
```

LINE (Line Number)	Both line and column numbers are relative to the upper left hand corner of the screen. The screen is itself positioned on the display device by the coordinates specified on the SCREEN statement.
COLUMN (Column Number)	
SIZE	Width of field display.
JUSTIFICATION	(Optional) Indicates whether the data in the field is to be displayed "RIGHT" justified or "LEFT" justified(can be abbreviated to "R" or "L"). If omitted the field is left justified.

Precise placement is usable with D, E, DR, ER, L and V fields. Placement specifications precede the query name (see Section 5.5.2 "Editable"), if both are used. Fields which use precise placement do not appear in the layout, although their data contents will appear on the actual screen display. Precise placement is most useful for

specifying the display of small fields whose names do not contain initial characters which are unique among field names, or to place related fields directly alongside each other.

Examples:

```
E MONTH [3,10,2]
E DAY [3,13,2]
E YEAR [3,16,2]
E AMOUNT [3,25,6,RIGHT]
```

The LINE and COLUMN coordinates for precise field placement may contain "addition" expressions that can be used with parameterization:[26]

```
E FLD [<Y>+5,<X>+10,8]
```

There can be only one '+' per coordinate; negative numbers and subtraction are not supported; and there must not be any embedded blanks in the coordinate expression. After the substitution of angle-bracketed parameters (see Section 5.17 "Parameterization") and/or the substitution of parameters in indirectly referenced (@@) files (see Section 1.3.3.1 "Passing Parameters in Indirect References"), the coordinate expressions must contain only constants (no field names).

Display width and justification can be specified by referring to the Data Dictionary element that describes a field. Use the "**W**%" and "**J**%" tokens to retrieve the display width and justification attributes from the Data Dictionary element for the field. E.g.:

```
E VENDOR [6,10,W%,J%]
```

5.6.1.1 Precise Placement of Text Blocks

TRANS can be instructed to display "text blocks" (see Section 5.6 "Screen Layout") using the following special precise placement syntax:

```
[LINE,COLUMN,HEIGHT,WIDTH]
```

LINE Line Number	(Same as regular precise placement)
COLUMN Column Number	(Same as regular precise placement)
HEIGHT	Number of lines to be displayed. Must be non-zero and positive.
WIDTH	Number of columns to be displayed.

For example, to display up to 6 lines of text stored in the TI field ADDRESS:

```
D ADDRESS [10,1,6,60]
```

It is the responsibility of the developer to ensure that text blocks do not overlap other items in the screen display.

26. This feature is useful when a standard block of fields is included by indirect reference (using @@) in several screens but must be placed at different locations in different screens.

5.6.2 Inclusive Field Names

A problem arises when there are two field names where one is included in the other as a substring. For example, "ST" and "STREET". When specifying a field for display in the layout section of a screen, a period (".") can be added to a field name to indicate that the user is specifying the **full** field name, and not a partial name. In the example, "----ST" is ambiguous as to whether the reference is to "ST" or "STREET", whereas "---ST." is specifically a reference to "ST" and "---STR" is a reference to "STREET". Where a reference is ambiguous, i.e., the partial field name is part of more than one actual field name, AdmScreen uses the first field name in the field names portion of the screen description that qualifies. Where possible, fields should be named so as to avoid ambiguity.

5.6.3 Displaying Fields More than Once

In a screen description, each of the field names can only be displayed once in the screen layout. If, in fact, the user wishes to display a particular field more than once on the screen, this can be done using virtual fields. However changing the data can only be done via the editable field. For example:

```
...
E OWNER
V XOWNER/A30 OWNER
SCREEN
OWNER: OWNER------------------------
...
MAIL TO: XOWNER-----------------------
END
```

5.7 Branches

The fifth component of the screen description is the BRANCHES paragraph, which identifies what "branches" can be called from the screen. "Branching" is when TRANS exits from the current screen and enters a new one (or re-enters the original one.) Branches can use data from the current record as key values to determine which record is to be displayed by the new screen.

The BRANCHES paragraph is separated from the screen layout by a line that begins with the word BRANCHES. The BRANCHES paragraph is terminated with the END statement, and may be followed by additional screen descriptions. (When there are no branches in a particular screen END terminates the screen description.)

The BRNC keystroke initiates the branching function; TRANS prompts for a branch name:

```
Branch to
```

If the user responds by entering a valid branch name, TRANS "branches" to the specified screen. Responding with another BRNC keystroke calls the "pop-up" Branch Menu, which displays all the branch choices available.

If implemented by the application developer, as described in Section 6.14 "HELP in TRANS", responding with the letter "H" will call TRANS HELP: on-line application-level assistance displayed in a pop-up window.[27]

Some common uses of this branching facility are as follows:

1. To change screens on the current displayed record. This type of branch would be used if a record contained too much information to be shown on one screen.
2. To display another record in the same file using data from the current record to identify the new record to be displayed. The new record is to be displayed via the current screen. For example, a person's record may identify the spouse's record, or children's record, and a branch might be to display the spouse's information using the same layout as the current record.
3. To display a record in another file using another screen. For example, a screen for a street index file keyed on street name and number may contain a branch which uses the account number to branch to a screen of the property ownership record (keyed by account number) at a particular street address.
4. Branching may also be used to call up another screen and/or another file that are not directly referenced from data on the current screen. This would be done to allow the user to access other files after concluding with the one on the current screen.

The branch section of the screen description contains the following for each branch.

```
BRANCH-NAME [TRO-NAME/]SCREEN-NAME [BRANCH-FIELDS]
BRANCH EXPLANATION TEXT
```

BRANCH-NAME is the name of the branch. Typing this name or the initial letters of this name after the BRNC key or while the Branch Menu is active tells TRANS to perform this branch. TRANS uses only the first two characters of BRANCH-NAME to distinguish between branch selections, so the branch names within a TRS should be unique in their first two characters.[28] (BRANCH-NAME "H" is usually reserved for Help in TRANS.)

TRO-NAME/SCREEN-NAME identifies the screen that should be used to display the "target" record of the branch. Branches may be made to screens within the current TRO (and described in the same TRS file) or to screens in another[29] TRO (and described in a different TRS file).

If the target screen is described in the same TRS file (and is consequently part of the same TRO) as the current screen only SCREEN-NAME, the name of the screen (the first element of the target screen's screen header line) is given. If the target screen is part of another TRO both the TRO-NAME (the target screen TRO name without the ".TRO" suffix) and the SCREEN-NAME must be provided, separated by a slash (/).

BRANCH-FIELDS are the (optional) field names of fields in the current record that form the key which identifies the target record to be displayed by this branch.

---

27. Using BRNC and "H" is not a branch to another screen, it is simply an alternative to the HELP keystroke for calling the TRANS HELP facility.
28. If "j" (lowercase) is included in the string assigned to the logical name option, one and two character BRANCH-NAMES may be used that share the same initial character. This option causes TRANS to wait for a terminator (carriage return) when the branch-name is entered by the user, allowing it to distinguish, for example, a BRANCH-NAME of "A" from a branch name of "AB".
29. There is a slightly higher overhead in branching outside a TRO file, because TRANS must open the new TRO file

If the target of a branch is another screen in the same TRO operating on the same record in the same file, the BRANCH-FIELDS specification may be replaced by the word "SAME". SAME indicates that the target screen should operate on the same record as the active screen. There are two subtle differences between branching using SAME and branching with explicit branch fields. "SAME" guarantees that the target screen branches to the same record, rather than to the first record in the target file with the key values of the branch fields. This is useful when the target screen operates on a file with multiple records with the same key values. Second, the use of BRANCH-FIELD "SAME" does not close and reopen the master file, which may have minor throughput benefits.

(If BRANCH-FIELDS are simply not present then when the particular branch is taken, the target screen will show the first record in the target file.)

BRANCH EXPLANATION TEXT is a short descriptive phrase that will appear in the entry for the branch in the Branch Menu.

In the following example the "A" branch is to the ACC screen in the ACCOUNTS.TRO file. The "F" branch is to the DESC screen in the FUND.TRO file in the [ACCTG] directory. The "V" branch is to the VENDOR screen in the current TRO.

```
A ACCOUNTS/ACC ACCT#
Examine detail accounts
F [ACCTG]FUND/DESC
Fund descriptions
V VENDOR VEND#
Vendor list
```

By default, the menu window is displayed centered at the bottom of the screen. The BRANCHES paragraph in the above example would display the following menu window:

```
 ----------------------------
|         Branch Menu        |
| A: Examine detail accounts|
| F: Fund descriptions       |
| V: Vendor list             |
|                            |
 ----------------------------
```

Select a branch in the menu by typing its branch code, or use the arrow keys to highlight the branch you want, then press SELECT.

If there are more branch entries than can be displayed in the menu window, the message "<more>" appears on the bottom line. Use PREV or NEXT to move forward and backward within the list of available branch options.

If a screen uses branches identical to those already specified in a preceding screen in the TRS file, then the BRANCH-NAME constitutes a sufficient description of the branch, i.e. the SCREEN-NAME, BRANCH-FIELDS and BRANCH EXPLANATION TEXT can be omitted and they will be picked up from the preceding description of the branch. The branch to "Examine detail accounts" in the previous example could be included in the next screen description in the same screen instruction file as follows:

```
BRANCHES
A
END
```

(You can back out of the BRANCH function via the HOME keystroke) See Section 6.9 "Branching and Subscreens" for details on how to use TRANS' branching facility.

5.7.1 Customizing the "Pop-up" Branch Menu

Instead of using the default menu heading and centered rectangle at the bottom of the screen, the developer can place the menu rectangle anywhere on the screen by giving the coordinates of the upper left corner, and optionally the rectangle size and a menu header text, on the BRANCHES line of the TRS using the following syntax:

```
BRANCHES line [column [#_of_lines [width]]] [Menu heading text]
```

For example:

```
BRANCHES 5 20 6 30 THIS IS A MENU
A ACCOUNTS/ACC ACCT#
Examine detail accounts
F [ACCTG]FUND/DESC
Fund descriptions
V VENDOR VEND#
Vendor list
END
```

will display a menu window, 6 lines long and 30 columns wide, starting at line 5, column 50:

```
 ----------------------------
|         Branch Menu        |
| A: Examine detail accounts|
| F: Fund descriptions       |
| V: Vendor list             |
 ----------------------------
```

Note that #_OF_LINES and WIDTH values both include the menu border; and #_OF_LINES includes the menu heading.

If any of the values are missing, TRANS will provide defaults. Since column, #_of_lines and width are positional, a zero may be present to get the default for any parameter, but still provide a fixed value for a subsequent parameter. E.g.

```
BRANCHES 16 40 0 30 MENU HEADING
```

will create a menu window starting at line 16, column 40, with a width of 30 columns. The number of branches present will determine the number of lines used.

```
BRANCHES 16 40 MENU HEADING
```

will create a menu window starting at line 16, column 40. The number of branches present will determine the number of lines used, and the longest branch text present will determine the width.

5.7.2 Automatic-only Branches

If two percent signs (%%) precede the text of a branch description, then that branch can only be activated automatically, as described in Section 16.2 "Automatic Branching: B$B and R$R".[30] Automatic-only branches are not presented to the user in the Branch Menu.

30. The AUTOBR subroutine (see Appendix H.13.1 "AUTOBR: Automatic Branch Control") provides even more flexibility in controlling branching. RMO calls to the AUTOBR subroutine allow automatic-only branches, either individually or universally, to be converted to manual branches (and back to automatic-only branches).

5.7.3 Calculated Branches

TRANS supports "calculated branches". To specify "calculated branches", the SCREEN-NAME for a branch is given in the TRS file as B$fieldname/XX, where the **SCREEN-NAME must start with the string "B$"** "fieldname" may be any alpha string, i.e. "B$BRANCH/XX" or "B$CALCBR/XX", and the **SCREEN-NAME must end with the string "/XX",**

Calculated branches work as follows: TRANS expects an alphanumeric (An) type field called "B$fieldname" (the same string as in the branches paragraph, less the "/XX") to exist in the TRO virtual record. TRANS expects this field to contain a TRO name followed by a slash and a screen name within the TRO (e.g. MENU/FIRST where MENU.TRO contains a screen called FIRST). The syntax for calculated branches is as follows:

```
BRANCH-NAME B$FIELDNAME/XX [BRANCH-FIELDS]
BRANCH EXPLANATION TEXT
```

For example, if we had the following lines in the TRS: TRANS will execute the branch as specified in B$fieldname. Using this feature sophisticated menu systems can be built where the[31] target screens for branching are kept in ADMINS files.

```
...
ER B$BRANCH/A24
...
BRANCHES
A B$BRANCH/XX
THIS IS A CALCULATED BRANCH
END
```

then if "[ACCTG]FUND/DESC" were entered into the field called B$BRANCH, the "A" branch would be to the DESC screen in the FUND.TRO in the directory [ACCTG]. The B$BRANCH field could be an actual field in the master file, a field linked from a link file, or a local field entered by the user or set by an RMO behind the screen.

5.7.4 SELFBRANCH: Self-branching with Multiple Index Files

When the main file for a screen has multiple indexes and any one of those indexes might possibly be the active index, doing a “self-branch” to another specific record using the same screen can be problematic - you need a way to tell TRANS which fields in the virtual record to use as the “branch keys” and the fields will be different depending on which index is currently active.

SELFBRANCH provides a solution. SELFBRANCH is a reserved branch-name in the BRANCHES section of the screen description that works with SELFBRANCH statements in the field declaration section of the screen description.

31. The B$KEYFIELDS RMO array is checked if no BRANCH-FIELDS are given for a calculated branch. The B$KEYFIELDS array allows you to set the keyfields (or key values) for a calculated branch at the time a branch is made. The B$KEYFIELDS facility, discussed in Section 16.20 “Calculated Branches with Variable Branch Keys”, allows you to develop and control even more complex screen families and menu systems.

To use SELFBRANCH, in the BRANCHES section, add

```
X SELFBRANCH
Any text
```

where 'X' is any branch letter, and 'Any text' is whichever branch text you want to supply.

In the screen's field declaration section you may add any number of SELFBRANCH statements, one for each of the main file's multiple indexes that might be active when a self-branch is required:

```
SELFBRANCH # [keyfield1 [keyfield2 […]]]
```

where '#' is any number between 0 and 9 indicating which index you are naming key fields for (0 refers to the primary index), and 'keyfieldn', etc. is a list of field names that should be used as key fields if a self-branch is requested when this index is active.

If no field names follow the 'SELFBRANCH #' a self-branch to top-of-file is performed. If no 'SELFBRANCH #' statement is present for the current active index, a self-branch on the current key values will be performed.

5.8 Time Card Entry Example

A screen is to be used to enter payroll time cards. The user enters the employee number, the regular hours for the week, and the overtime hours for the week. The screen checks via a link that the employee exists in the employee master file and displays the employee name on the screen. The screen also checks that the regular hours do not exceed 40, that overtime hours are only entered when regular hours equal 40, displays a message for employees with less than 40 hours, and shows the total hours worked as well. The DEF of TIME.MAS could be as follows.

```
*    TIME.DEF
MAS 1000
EMPL# X9999 KEY1      "employee number"
REGH D2               "regular hours"
OVTH D2               "overtime hours"
```

The DEF of EMPL.MAS (the employee master file) might contain the following.

```
MAS 1000
EMPL# X9999 KEY1      "employee number"
LNAME A20             "last name"
FNAME A10             "first name"
HRLYRT D2             "hourly rate"
RHTD D2               "regular hours to date"
OHTD D2               "overtime hours to date"
PTD D2                "total paid to date"
TITLE A20             "job title"
ADDR A20              "home address"
CITYST A20            "and city state"
```

The screen description would look as follows.

```
*    Screen runs on TIME.MAS
TIME TIME.MAS 1 APPEND
*
*    Use LINK to get first and last name
*

LINK EMPL.MAS
KC EMPL#
L LNAME
L FNAME
END
*
*  Fields for display
*
E EMPL#
* Display the name
D FNAME
D LNAME
* Check for correct employee number
C EMPL# NE 0000 AND LNAME EQ ' '
INCORRECT EMPLOYEE NUMBER
E REGH
* Check regular hours not greater than 40
C REGH GT 40.00
REGULAR HOURS EXCEEDS 40
E OVTH
* if overtime, regular must be 40
C REGH NE 40.00 AND OVTH NE 0
REGULAR HOURS ARE NOT 40
* Compute total hours
```

```
V TOTH/DP REGH + OVTH
SCREEN
CE TIME CARD ENTRY SCREEN
BL
EMPL#: EMP-            NAME: FNAME----- LNAME--------------
BL
REGULAR: ----REGH      OVERTIME: ---OVTH
BL
                             TOTAL HOURS: -----TOTH
END
```

A branch is added to the screen that would display under the time card entry screen other information from the employee master file record. First, add a display rectangle to the SCREEN statement for the time card.

```
SCREEN 1 1 8 80
```

And then place the following 3 lines just before the END statement.

```
BRANCHES
E EMPL EMPL#
DISPLAY EMPLOYEE INFORMATION
```

And finally place a second screen description, starting with its header line, after the END statement of the first screen description.

```
EMPL EMPL.MAS 1
E EMPL#
D FNAME
D LNAME
D TITLE
D ADDR
D CITYST
D HRLYRT
D RHTD
D OHTD
D PTD
* place the display rectangle at
* row 9, column 1, for 8 lines, and 80 columns
SCREEN 9 1 8 80
EMP-  FNAME----- LNAME--------------   TITLE--------------
      ADDR----------------
      CITYST--------------
BL
HOURLY RATE:    -----HRLY
REG. TO DATE:   -----RHTD
OVT. TO DATE:   -----OHTD
PAID TO DATE:   ------PTD
BRANCHES
T TIME EMPL#
RETURN TO TIME ENTRY SCREEN
END
```

NOTE

We redisplay the employee number and name on the employee master screen. This is done because the user may move around among employee records in the employee master file, and should see the name for the displayed employee number.[32]

Also included in the second screen is a branch back to the first screen.

TRANS will erase the bottom half of the screen when the user takes the T branch back to the TIME screen if the display rectangle set for the time entry screen is the whole screen, i.e. the SCREEN statement is used without screen layout coordinates.

32. Key fields are never "edited". They can only be changed via a record transfer, described in Section 6.7 “Record Operations”. TRANS always interprets entry into a key field as a search request to go and find the record with the entered key.

5.9 Multi-Record Screens

On a multi-record screen the contents of several records are displayed simultaneously on the screen. This is controlled by the RPS, records per screen, keyword on the screen header line, being set to more than "1" (see Section 5.3 "Screen Header Line").

The following restrictions apply to multi-record screens.

1. By default, only one line of information is repeated per record. The heading, which is considered to be everything but the last line of the layout, may contain data fields as well as literal text. (Data fields in the heading must appear **first** in the field names section of the multi-record screen.) The data fields in the heading will be taken from the **first record** that appears on the screen. The last line of the layout, i.e. the repeating fields part, should contain data fields only[33] (no literal text). This last line is displayed repeatedly for each record on the screen.

   If the value of the RPS keyword is immediately followed by "/n", e.g. 6/3, then AdmScreen is instructed that the last "n" lines of the screen layout repeat for each record to be displayed. That is, by appending the "/n", more than one line of data can be displayed for each record in a multi-record screen. Here too, only data fields and not literal text can be contained in the repeating portion of the multi-record display. Text placed in virtual alphanumeric fields may, however, be included in the multi-record display.

   Up to 15 lines may be repeated for each record, e.g: 4/15 to display 4 records per screen and 15 lines per record.
2. Multi-record screens may be used for display and for update. However, the APND, INS, DEL, or TRF keystrokes cannot be used with multi-record screens. (See Section 6.7 "Record Operations" on TRANS for an explanation of these keystrokes).
3. **Local**[34] ER fields should **not** be used in the repeating portion of a multi-record screen. (ER fields are OK in the heading portion of the screen.)
4. NOWRITE should **not** be used with multi-record screens.
5. A Multi-record screen may not be used in conjunction with LFEXIT control. Therefore, a multi-record screen is incompatible with the keywords LFEXIT or LFBACK, the REQUIRE statement, or the CLF Check statement.
6. Subscreens are not available in multi-record screens.

One use for multi-record screens is to display a few fields from several records, with a branch to a screen which displays more fields for a single record. The UP and DOWN (arrows) keystrokes can be used to move the cursor from record to record on the page of a multi-record screen, to single out particular records.

---

33. If it is necessary to precisely place every field on the repeating line or lines, put a 'BL' (blank line) on those lines in the screen layout.
34. Main file and linked-in fields that are referenced in the TRS as ER fields (because they might be changed by the RMO) may be used (and commonly are used) in the repeating portion of a multirecord screen.

A second use for multi-record screens is similar to showing a little about many records and then branching to a full screen about one record, only the multi-record screen is displaying an index file and the branch is to the master file. (The LINK statement on the index file screen can be used to include some information from the master record on the multi-record index screen.)

A third use for the multi-record screen is in those cases where there is a file of master records and a file of related repeating detail records per master record. For example, the accounting ledger record and the detailed purchase order records for each account. Then one could display a ledger record for a particular account and branch to a split screen that displayed several of the detail purchase order records for the same account. The BREAK keyword (see Section 5.3.1.5 "BREAK On A Multi-Record Screen") would be used in this situation to restrict the multi-record display to records for the same account only.

5.9.1 Multi-Record Screen Example

The following is an example of a multi-record screen with a single line of repeating information for each record.

```
DIREC DIREC.MAS 10
E LNAME
E FNAME
E ADDRESS
E TELNO
SCREEN
CE DIRECTORY LISTING
BL
NAME                          ADDRESS         TELNO
LNAME--------- FNAME------  ADDR---------   TEL----
END
```

This screen would display 10 consecutive telephone directory records at a time. The example in Section 5.9.2.1 "BREAK Example" shows the use of data in the heading.

In the following example of a multi-line, multi-record screen, there are 4 lines of data for each of the 5 records displayed on the screen. The virtual fields DASH1 and DASH2 are used to place literal text in the repeating section of the screen .

```
*
MAILING CUSTOMER.MAS 5/4 NOMSG
E CUSTCODE
*
D CUSTNAME
D ADDR1
D ADDR2
D CITY
D STATE
D ZIP
V DASH1/A30 '------------------------------'
V DASH2/A30 '------------------------------'
SCREEN
CE MAILING LIST
CODE AND NAME                 ADDRESS
BL
CUSTCODE-                     ADDR1------------------------
CUSTNAME--------------------- ADDR2------------------------
                              CITY------------ STATE-  ZIP--
DASH1------------------------ DASH2------------------------
END
```

Below is a sample of the screen output produced from the instruction file above.

```
                    MAILING LIST
NUMBER AND NAME                ADDRESS

B217104                        Room 30
BAY COLONY PAPER COMPANY       450 Broadway
                               Danvers        MA      01920
------------------------------ ------------------------------
C217535                        2225 Lowell Street
CAMBELL & MORRISON PAPER, INC.
                               Chelmsford     MA      01824
------------------------------ ------------------------------
F217654                        2nd Floor
FAY PAPER PRODUCTS             332-22 Van Ness Street
                               Boston         MA      02100
------------------------------ ------------------------------
F217885                        1254 Washington Street
FRANKLIN & WALSH PAPER SUPPLY
                               Worcester      MA      01613
------------------------------ ------------------------------
P217231                        Suite 14-5
THE PAPER TREE                 91 West Seventh Street
                               Lowell         MA      01853
------------------------------ ------------------------------
```

5.9.2 BREAK In a Multi-Record Screen

The BREAK keyword placed on the header line of a multi-record screen description instructs TRANS to display only records with the same (full or partial) key on a particular multi-record display.

Two keystrokes described in Section 6.5 “Field Logging” have special functions for use in multi-record screens with BREAK . The PRBK keystroke moves to the previous control break rather than the previous sequential page. Likewise, the NBRK keystroke will start the next display page at the next control break, rather than the next sequential page. The NEXT, PREV and NREC keystrokes always operate on sequential pages of records without regard to the BREAK feature.

After the BREAK keyword, or after the key name which follows the BREAK keyword, a partial key field designator can be used. For example, if the active file has one A10 key field, you could specify "BREAK =XXXX" or "BREAK KEYNAME =XXXX" for a control break whenever any of the first four characters of the key field changes.

Syntax for partial field BREAKs are the same as in REPORT (see Section 7.7.5 “Break At Partial Field” for details):

=X...	for an alpha (A) field
=A...9...	for picture (X) fields
=YY or =YY-MMM	to break on the year or year and month in a standard-format date (DA) field.
=.SUBFIELD	to break on the SUBFIELD, e.g. if the key is ACCOUNT and it has a subfield DEPT then: BREAK =.DEPT or BREAK ACCOUNT = .DEPT

When a screen uses BREAK , only alternate indices that contain the BREAK field as one of the key fields will be shown in the File/Alternate Indices menu.

5.9.2.1 BREAK Example

```
*          DEPT.DEF
TAB 100
DEPT X99 KEY1  "department code"
NAME A20       "department name"


*          BUDGET.DEF
MAS 1000
DEPT X99 KEY1  "department code"
OBJ X999 KEY2  "object of expense code"
AMT D2         "budgeted amount"
```

The multi-record screen follows:

```
AMT BUDGET.MAS 10 NOMSG BREAK DEPT AUTOCR
*
*     Link to get the department name
*
LINK DEPT.TAB
KC DEPT
L NAME
END
D NAME
E DEPT
E OBJ
E AMT
SCREEN
CE DISPLAY BUDGET AMOUNTS FOR DEPT NAME----------------
BL
                          DEPT      OBJECT      AMOUNT
BL
                           D-        OB-    ---------AMT
END
```

The screen would break at the end of each department even though BUDGET.MAS is keyed on department and object. The BREAK keyword can be followed by the field name of one of the keys, instructing TRANS to "break" on that key field. The following is a sample of the display generated by the screen description above:

```
DISPLAY BUDGETS AMOUNTS FOR DEPT PUBLIC SAFETY

      DEPT      OBJECT      AMOUNT

      05        101      101,765.00
      05        102       20,378.00
      05        103        3,234.00
      05        104       10,250.00
      05        201       30,788.00
      05        202          376.00
       05        203        1,450.00
```

Note, that although up to 10 records are **allowed** on the screen, there are only 7 records in Department 05, and hence only 7 records are displayed.

5.9.3 NOMULREC: Allow non-repeating items in multirecord layout

The NOMULREC statement, identifies a list of non-repeating fields that can appear in the multi-record portion of the screen.

```
NOMULREC fieldname [fieldname …]
```

Place the NOMULREC statement in the field declaration section of the the TRS.

5.10 Video Highlighting Facilities

The setting of the logical name OPTION (see Appendix A: "Options") can be used to create general highlighting effects in TRANS.

If the logical name OPTION includes the letter "R", TRANS will display data in reverse video with the keys underlined and display literal text in normal video. This is the same method TRANS uses when in the GENED Mode (see Section 6.16 "General Editor Mode ("GENED")").

If the logical name OPTION includes the letter "W", TRANS will display keys in reverse video and other data in increased brightness (bold).

Video highlighting can be set on a field by field basis, by placing video attributes keywords after the field name in the TRS fields section.

The video attributes keywords are:

```
BOLD
UNDERLINE or UL
BLINK
REVERSE
FULL
NORMAL
```

All codes can be abbreviated to two or more characters.

"FULL" instructs TRANS to always highlight the full width of the field, regardless of how many positions are actually filled with data. Attributes applied with "FULL" persist even when the field is cleared for editing. "NORMAL" instructs TRANS to display the field in normal video when a more general video instruction (described below) would otherwise cause it to be highlighted.

The video attributes keywords are preceded by a "%" (percent sign). Two or more attributes can be combined with a "+" (plus) character (no imbedded blanks):

```
%UL+BOLD+FULL
```

The video attributes keywords must be the last item in the field declaration statement, unless the field has a LOOKUP file (see Section 5.11 "LOOKUP Window") associated with it. Examples:

```
E NAME  %BOLD+FULL

ER ZIP/X99999 [6,10] %UL %LOOKUP ZIP.TAB
```

The **VIDEO** statement may be used in the TRS to assign default video attributes to all fields in four different classes. The syntax is:

```
VIDEO CLASS video_attributes [CLASS video_attributes ...]
```

Where CLASS may be KEYS, EDIT, DISP, or MULTI. "KEYS" defines default video attributes for all key fields; "EDIT" defines default video attributes for all editable fields (including loggable fields); "DISP" defines default video attributes for all display fields; and MULTI defines video attributes for the active record in multi-record screens[35]. Video attributes are specified in the VIDEO statement using the same keywords and syntax as described above (the "%" delimiter is only used on the field declaration line). Example:

```
VIDEO KEYS BOLD+REVERSE+FULL EDIT REVERSE+FULL DISP BOLD
```

MULTI attributes are combined with attributes specified for classes of fields. E.g.:

```
VIDEO MULTI REV KEYS UL
```

will display key fields underlined in all the records of a multi-record screen and display the entire active record in reverse video.

The VIDEO statement must be the first line following the SCREEN header line. The "VIDEO" statement effects only the screen where it appears.

VIDEO attributes specified for individual fields override those specified in a video statement. For example, the NORMAL keyword in the field declaration statement could be used to turn off the video attributes for a specific field, as follows:

```
ITEMLIST INVENTORY.MAS 10 NOMSG 132
VIDEO KEYS BOLD DISPLAY REVERSE
     .
     .
     .
DR TODAY/DA %NORMAL
     .
```

To avoid having to put the VIDEO statement into all the screens if you want all key, editable, and display fields to be highlighted the same way throughout an application or entire system, you may assign the content of the VIDEO statement to the logical name ADM$SCR_VIDEO before SCREENing the TRS's. When ADM$SCR_VIDEO has a value assigned to it, AdmScreen treats that value as the VIDEO statement for every screen it compiles.

For example, the following logical name assignment:

```
$ assign "KEYS BOLD+FULL EDIT REV+FULL DISP UL" ADM$SCR_VIDEO
```

would cause the screen to be compiled with all key fields to be displayed in bold for the full display length of the fields, all editable fields other than key fields to display in reverse video for the full length of the field, and all display only fields to be underlined only for the part actually containing data.

35. MULTI highlights the entire active record area, not just the fields, giving the screen an appearance similar to a LOOKUP window. (For MULTI the FULL attribute is redundant.)

To provide even greater flexibility, TRANS video attributes can be changed at run time without re-screening; either globally or differently for different users according to individual preferences.[36] ADM$TRANS_VIDEO has exactly the same syntax as ADM$SCR_VIDEO, described above. ADM$TRANS_VIDEO does not override video settings which are compiled into the TRO or the RMO (see precedence discussion below). ADM$TRANS_VIDEO is evaluated after the first pair of BEGREC (or MULREC) calls when a screen is first entered. The RMO can assign the ADM$TRANS_VIDEO logical name at the first calls (BEGREC or MULREC) in a screen and the video attributes specified will apply for that screen.[37]

The KEY_VIDEO statement is used in the TRS to modify the video highlighting action of TRANS resulting from the logical name ADM$TRANS_VIDEO, e.g. the statement

```
KEY_VIDEO AFIELD BFIELD
```

would cause the fields AFIELD and BFIELD to be displayed with the same video attributes as specified for the screen's key fields in the logical name ADM$TRANS_VIDEO.

There are several distinct mechanisms in ADMINS for specifying the video attributes of a field displayed on a screen. The complete order of precedence is, from low to high:

```
(lowest)  R and W OPTION settings (see Appendix A: "Options")
          ADM$TRANS_VIDEO logical name
          ADM$SCR_VIDEO logical name
          VIDEO statement
          video attribute keywords in field declarations
(highest) H$CODE settings (see Section 16.5 "Highlighting Fields")
```

36. ADM$TRANS_VIDEO works only with TROs, not in GENED mode.
37. See Section 15.2.1 "Beginning of Record Processing: S$S = 'BEGREC'" for a general discussion of Beginning of Record RMO processing (BEGREC). See Section 16.22 "Multi-Record RMO Support" for a discussion of RMO communication with multi-record screens (MULREC calls). See Appendix H.9.1 "CRLOG - Create or Delete a Logical Name" for description of the CRLOG subroutine, which could be used to assign the ADM$TRANS_VIDEO logical name when the screen is entered.

5.11 LOOKUP Window

TRANS' LOOKUP windows provide quick, easy access to tables that contain information relating to editable and displayed fields. When the TRANS cursor goes to a field that has a LOOKUP file associated with it, the user can call up a window with a scrollable multi-record display and select a value from the display for automatic entry. TRANS treats the selected entry exactly the same as if the value had been typed via the keyboard.[38]

LOOKUP windows can be designed to start displaying records at a designated point (a value for the first key) in the LOOKUP file, or to display only a subset of the LOOKUP file, defined either by a key range or by a select expression. LOOKUP windows may also link in values from an additional file, and/or transfer values from any field in the file to fields other than the field.

LOOKUP can be used for any editable or display (E, D, L, DL, ER, DR, LR) field in the screen.

LOOKUP syntax is:

```
E FIELD %LOOKUP file_name or ADD
  [ HEADING LITERAL ... ]
  [ TITLE [%VIDEO] LITERAL ]
  [ FOOTING [%VIDEO] LITERAL ]
  [ DETAILBREAK K_FIELD ]
  DISPLAY D_FIELD[/WIDTH] ...
  [ KEY  K_FIELD1 [K_FIELD2]... ]
  [ KEY_RANGE LO_FLD1 [LO_FLD2...] HI_FLD1 [HI_FLD2...]
  [ PROMPT PROMPT_TEXT/1 [PROMPT_TEXT/2...]
  [ RETURN  FIELD[/NOTYPE] ]
[ WINDOW LINE COL #LINE #COL ]
  [ LINK =prefix link_file KEY[S] IS/ARE L_KFLD1 L_KFLD2...]
  [ TRANSFER LKUP_FIELD INTO SCREEN_FIELD ]
  [ SELECT expression ]
  [ CREATE FIELDNAME/TYPE expression]
  [ NOLOCKEDRECORDS ]
  [ SEARCHBUTTON|NOSEARCHBUTTON]
  [ BREAK KEY_FIELD ]
  [ TOTAL K_FIELD ]
  [ SUMMARY FIELD|"literal"[/OP][/ALIGN=FIELD][/WIDTH]
  [ CAPS ]
  [ BOUND ]
  [ BOUND_KEY ]
  [ CR_EXIT ]
  [ LK_CLOSE ]
  [ PUSHBUTTON ]
  [ IDENT NAME ]
```

Use

%LOOKUP file_name

to specify a LOOKUP window when the window is to be based on a ADMINS data file that is being used as a table.

Use

%LOOKUP ADD

38. For display-only fields (D, DR, DL) LOOKUP windows are also display-only, e.g. no value can be selected or written.

to modify the Data Dictionary-generated automatic LOOKUP window for a field that is bound to a Codelist Table (see Appendix I.7.3 "Automatic Lookup Windows "). The statements described below will **override** the specification for that characteristic in the Dictionary[39]. The %LOOKUP ADD syntax is also used when the Data Dictionary-generated automatic LOOKUP window is to be used with a display-only field (without the %LOOKUP ADD the cursor would not go to a display-only field).

LOOKUP sub-statements must be indented. Up to 7 LINK sub-statements and/or up to 25 TRANSFER sub-statements may be used in a single LOOKUP paragraph. All other sub-statements may appear only once. The SELECT, KEY, KEY_RANGE, and PROMPT sub-statements may be continued to a new line using the ":" continuation operator.

Sub-statement	Description
DISPLAY	Lists the fields in the LOOKUP file to be displayed in the window. **DISPLAY is the only required sub-statement in a LOOKUP paragraph.**
HEADING	Must contain one alphanumeric literal (no spaces) for each field named in the DISPLAY statement. (If no HEADING statement is present, the DISPLAY field names are used as headings). Heading text is justified according to the data type of the corresponding DISPLAY field.

TITLE	Put literal text on the top line of the LOOKUP window. The text will be centered in the window.
	The TITLE can contain information from fields in the virtual record. Use the field name surrounded by "%", for example:
	`TITLE Lookup for %TBLNAME% table`
	or
	`TITLE %LOOKUPTITLE%`

FOOTING	Put literal text on the bottom line of the LOOKUP window. The text will be centered in the window. Video attributes for FOOTINGs are specified as described above for TITLE.
DETAILBREAK	Accumulate values until there is a change in the specified key value (similar to TOTAL *keyfield* in REPORT). DISPLAY statement will display accumulated values for each break. The fields on the DISPLAY line will have implicit or explicit aggregation operators similar to the fields on the TOTAL line in REPORT. The default (implicit) aggregation operators are /V for numeric fields, and /FI for non-numeric fields and key fields (regardless of type)
KEY	Start to display records at the specified key value. Up to 9 key values may be specified, corresponding to the key fields of the LOOKUP file. K_FIELD1 through K_FIELDn must be defined in the screen's virtual record.

39. When you use %LOOKUP ADD the names of the code, description and user action fields in the codelist table are {D}CODE, {D}DESCR, and {D}UAC, respectively. Use these names to refer to these items in any LOOKUP substatements, e.g.: SELECT {D}CODE NE '00000'.

Sub-statement	Description
KEY_RANGE	Limits the LOOKUP display to records with a key value within the range of the values defined by the LO_FLDn and HI_FLDn fields. Up to nine low/high pairs may be entered (use ":" line continuation if necessary). The number of lower limit fields given must be equal to the number of upper limit fields given.
PROMPT	Allows the user to customize the key field prompts that appear when the user clicks ***Find***. LOOKUP will prompt, using the specified string in place of the field name, for each key value specified in the PROMPT sub-statement. PROMPT syntax is: `PROMPT PROMPT_TEXT/1 [... PROMPT_TEXT/9]` The prompt string may be up to 20 characters long. If any imbedded blanks appear in the prompt string the entire string must be enclosed with apostrophes. The prompt string is followed by a slash and the associated key number, i.e. "/1" appended to the prompt string identifies that string as the prompt for KEY1, and "/9" identifies the prompt string for KEY9. An example: If KEY1 of the LOOKUP file, EMPLOYEE.MAS, is EMPNO, the employee's ID number, then by default LOOKUP will prompt as follows when you click ***Find***: **`Find EMPNO beginning with`**[a]**`:`** If the following PROMPT sub-statement is introduced into the LOOKUP paragraph: **`PROMPT Employee_ID/1`** then LOOKUP will prompt as follows: **`Find Employee_ID beginning with:`** Prompt strings must appear in key order ([D]KEY1 first), although keys can be omitted. Values for omitted keys are taken from the current record in the LOOKUP window. If no PROMPT sub-statement is specified for a LOOKUP paragraph, LOOKUP prompts for [D]KEY1 only by name, and does a partial key search for the value of [D]KEY1 given in response.
RETURN	Identifies the field in the LOOKUP file whose value should be put into the editable field when a selection is made. If no RETURN statement is specified the CANCEL button will be the default button when the LOOKUP window is displayed.
WINDOW	If a LOOKUP paragraph has a `WINDOW line# column# 0 0` statement the dialog box will appear with the upper left corner at the specified line and column. The last two values must be 0 (zero) for AdmTRANS to utilize the WINDOW statement (otherwise it is ignored)

Sub-statement	Description

LINK	Allows information to be brought in from additional files. The key fields, L_KFLDn, used to perform the link may be fields in the LOOKUP file itself, or be fields linked into the window via previous LINKS (identified with their prefix, as described below). The **=prefix** is put in front of all field names in the linked file, to automatically rename them for the scope of the LOOKUP paragraph. This prefix must be present when any field[b] from the LINK file is subsequently referenced in the LOOKUP paragraph, e.g.:
	`LINK =CS_ CUSTOMER.MAS`
	
	`SELECT CS_D%SALESREP EQ 'Cosmo Carducci'`

TRANSFER	Used to transfer a value from a file field (LKUP_FIELD) into a field (SCREEN_FIELD) other than the field[c] SCREEN_FIELD may be a field in the screen file, or an ER or DR field defined locally for the screen. The data type of LKUP_FIELD and SCREEN_FIELD must be the same (although the length might differ). **Note that TRANSFER sub-statements only transfer values when a record is selected in the LOOKUP window.** If data is typed by a user directly into the field with the LOOKUP paragraph, no TRANSFER takes place. The BOUND keyword, described below, provides a means to insure that specified transfers will take place.

SELECT	Displays only those records in the LOOKUP file that satisfy the SELECT expression. Fields in the LOOKUP file, LOOKUP LINK files, fields in the virtual record of the screen, or constants, may be used to form the SELECT expression. When a field from the virtual record in the screen (i.e. outside the LOOKUP paragraph) is referenced, prefix[d] the field name with a ~ (tilde), e.g.
	`SELECT LKUPFLD EQ ~SCRFLD`
	where LKUPFLD is a field from the LOOKUP file, and SCRFLD is a field from the screen's virtual record, or
	`SELECT ~D%OCC_CODE EQ 'LABORER'`
	where D%OCC_CODE is the codelist description for OCC_CODE, a field in the virtual record.

CREATE	LOOKUP in TRANS supports "virtual fields" for display in the LOOKUP window. The general syntax is: `CREATE FIELDNAME/TYPE expression` i.e. similar to CREATE statements in REPORT and V statements in TRANS. These fields are for display only, their values are only calculated after a record has been selected for display.
NOLOCKED-RECORDS	Do not display (select) records that are currently locked by another user.
SEARCHBUTTON **NOSEARCHBUTTON**	Add ***Search*** button (finds requested string at any positon in key value[e]. NOSEARCHBUTTON disables Search for a particlar lookup when Search is enabled via TRANS$ENV

Sub-statement	Description
BREAK	If the BREAK keyword followed by a key field name is included in the LOOKUP paragraph, the LOOKUP window will "break" on that key, i.e. only records with the same full or partial key value will be displayed together. This facility performs in a manner similar[f] to BREAK in a multi-record screen (see Section 5.9.2 "BREAK In a Multi-Record Screen").
TOTAL	Specifies key field break for totaling - used with SUMMARY statement

SUMMARY	Uses break key field specified in TOTAL statement.
	The SUMMARY statement is similar to the DISPLAY statement, - it's used to specify which fields to display.
	As with DETAILBREAK, the fields on the SUMMARY line will have implicit or explicit aggregation operators similar to the fields on the TOTAL line in REPORT. The default (implicit) aggregation operators are /V for numeric fields, and /FI for non-numeric fields and key fields (regardless of type)
	In addition, fields on the SUMMARY line may have an /align=field qualifier, instructing to align the field with the specified field on the DISPLAY line.
	SUMMARY may contain literals. Literals must start and end with apostrophes, and may be followed by an /align qualifier.
	
	For example,
	
	`er getempno/x9999 % ypafth.mas`
	` heading EmpNumbr Code Hours GrossPay`
	` display empno hc hrs/10 gross`
	` total empno`
	` summary empno 'Total Pay'/align=hrs gross`
	
	Only one TOTAL/SUMMARY pair may be present in a

CAPS	If you press the LOOK key while LOOKUP is active, LOOKUP prompts for a key value to search for. If the CAPS keyword is used in the LOOKUP window specification, your response is converted to upper case before the search takes place.
BOUND	Prevents a user from entering into the field except by selecting from the LOOKUP window. When BOUND is specified in a LOOKUP paragraph, LOOKUP is called immediately if any attempt is made to type in the field. BOUND is especially useful when TRANSFERs are specified for the LOOKUP paragraph, to insure that the intended data transfers take place. BOUND must be explicitly specified for each field that is associated with a particular LOOKUP paragraph (using the "%LOOKUP =NAME" syntax). See the discussion of the IDENT keyword below.

Sub-statement	Description
BOUND_KEY	Like BOUND, BOUND_KEY prevents a user from altering the field except by selecting from the LOOKUP window. BOUND_KEY, however, does allow typing into the field. With BOUND_KEY in effect, whatever is typed into the field is used as a key specification for the LOOKUP window, which displays automatically when the entry is terminated. LOOKUP starts displaying records at the point in the file that is the best match (value of the first key) for the value entered. As with BOUND, when BOUND_KEY is in effect the only way to actually change the value of a field is by selecting a value via the LOOKUP window.
CR_EXIT	Moves the cursor to the next field, and, if the local RMO field F$UNCKEY[g] is present, calls the RMO with F$UNCKEY set to "LRET", when the LOOKUP window is cleared (via HOME) without selecting a value. If CR_EXIT is not specified, the cursor remains at the same field, and the RMO is not called, if LOOKUP is cleared without selecting a value.
LK_CLOSE	LK_CLOSE instructs TRANS to close the data files opened by the LOOKUP window when it leaves LOOKUP. Without LK_CLOSE, files opened by a LOOKUP window remain open until another LOOKUP window opens different files or until TRANS exits the current screen. Leaving the LOOKUP files open is more efficient if it is likely that the next use of LOOKUP would use the same files. Use LK_CLOSE when this assumption is not valid or whenever you want to make sure the files are closed immediately when TRANS leaves LOOKUP. LK_CLOSE can be especially useful to conserve resources when complex screens must operate near TRANS' limits.

Sub-statement	Description
PUSHBUTTON	USe PUSHBUTTON to add an additional pushbutton to the row of buttons at the bottom of the Lookup dialog box. The syntax is: `PUSHBUTTON LabelText ACTION=action(s)` where:
Label Text	is the text that appears on the button. Use apostrophes to enclose a label that contains embedded blanks.
ACTION	Specify the button's action using the same TRANS keystroke macro syntax used for regular push buttons (see Section 5.5.21.1 "PushButton Object"), macro definitions in the TRANS$ENV file (see Section 6.17.2 "Define Macro Function"), and in the alphanumeric argument method for the setkey routine (see Appendix H.13.14 "SETKEY - Simulate Keystrokes in TRANS") For example, this button may be used to signal to the RMO to take some special action when returning from the dialog: PUSHBUTTON 'New Customer' M$M_11 would call the RMO with M$M set to '11' when the dialog box closes. The RMO might then branch to a special screen for adding new customers. By default, the Lookup window is always cancelled (e.g. no record is selected and no value is returned or transferred) before the PUSHBUTTON's action occurs.
&TRANSFER	To TRANSFER fields the last action for the PUSHBUTTON can be a single &TRANSFER statement. The syntax is: `&TRANSFER sourcefield INTO targetfield` Observe that the ampersand (&) in front of the TRANSFER verb is part of the syntax.
&OK	To RETURN and TRANSFER fields (the same result as clicking the OK button) use "&OK" as the last action for the PUSHBUTTON `PUSHBUTTON 'Buttontext' M$M_nn &OK` When this button is pressed all transfer and return statements are executed, and the RMO will get the usual 'UX' and 'UP' calls, but F$UNCKEY will be set to 'nn' instead of 'LKUP', where '01' <= 'nn' <= '99', thus making it possible for the RMO to know which button was pressed

Sub-statement	Description
IDENT	To avoid having to define identical LOOKUP clauses for each field in a screen where you have several fields of the same type, looking up in the same table, and displaying and returning the same fields, you may name the LOOKUP clause for the first field, e.g. IDENT NAME. NAME is a character string constant up to 4 characters long (if longer, only the first 4 characters are used). On subsequent fields in the screen, where you want to invoke the same LOOKUP window, refer to the named LOOKUP clause. The syntax is: `E FIELD %LOOKUP =NAME [BOUND]` There can be no spaces between the equal sign and the name. BOUND must be explicitly specified for each use of the same LOOKUP paragraph. When specifying BOUND for a LOOKUP window that references a previously specified LOOKUP paragraph, BOUND must appear on the same line as "%LOOKUP =NAME".

a.Find will prompt "**Find <fieldname or descriptor> with value:**" if the field type is numeric or a date

b. The =prefix precedes the D% or U% prefix when the codelist description or user action code (see Appendix I.7.2.1 "Update Internal Codelist Tables ") for a field from the link is being referenced, i.e.
SELECT CS_D%SALESREP EQ 'Cosmo Carducci'

c. The RETURN statement is used to put values into the field.

d. The ~ (tilde) prefix is used both to signal that the field is in the screen virtual record, rather than the LOOKUP file, and to rename the field in case the LOOKUP file has a field with the same name. The tilde prefix precedes D% or U% when the codelist description or user action code for a field in the virtual record is being referenced.

e.See Section 6.11 "Lookup Windows"

f. The partial key designator syntax (=YY-MMM etc.) is not supported in LOOKUP.

g. See Section 16.16 "Subscreen Status and Control: ADM$SUBSCR"

If you do not want to show the whole length of a DISPLAY field, indicate the desired display length "n" by adding "/n" to the field name, e.g. to show the first 10 characters of the A40 field CITY, the syntax is CITY/10.

If a LOOKUP window is available for a field on the screen, pressing LOOK when the cursor is at that field activates the LOOKUP window.[40]

If LOOK is pressed before anything is typed into the field, TRANS displays the table file starting at the top of the file (unless a KEY or a KEY_RANGE statement is present). If anything[41] is typed into the field before pressing LOOK, TRANS uses that as a key value in the LOOKUP file, and displays records starting with that key. If a KEY_RANGE statement "locks" the LOOKUP window into a specific key value, then the value is used for the next key. For example, if a LOOKUP file has two keys

40.If the LKDO keystroke is used the LOOKUP window is launched in display-only mode, as described in Section 6.11.1 "Display-only LOOKUP"

41. A key value for the LOOKUP table can be entered, regardless of whether the field on the screen has the same data type as the LOOKUP table key. For example, if the LOOKUP clause returns an account number, type X9999, that is looked up via the account holders last name, you could type the first few characters of the last name (alpha characters) into the editable account number field (all numerics), then press LOOK, to "home in" on the part of the LOOKUP file you want.

```
FISCALYEAR X9999 KEY1
DEPT X999 KEY2
```

and the LOOKUP paragraph had this KEY_RANGE statement:

```
KEY_RANGE FISCALYEAR THISYR THISYR
```

then the value typed into the field before the pressing LOOK would be used for DEPT, and the LOOKUP display would start with that record.

LOOKUP behaves in a special way if the statement

```
create LKUP$LCK/I
```

is present in the LOOKUP paragraph. AdmTrans will set LKUP$LCK to 1 if a record is locked, or to 0 if the record is not locked by another user. This behavior makes it possible to include a statement like e.g.:

```
create lck/a20 if lkup$lck eq 1 then 'Record Locked' else ' ' end
```

in the LOOKUP paragraph, and then include teh field LCK in the DISPLAY statement, so that the LOOKUP display will indicate if a record being shown is currently locked by another user.

See Section 6.11 “Lookup Windows” for details on navigating inside the LOOKUP window.

The user selects a value in LOOKUP by positioning the LOOKUP's cursor at the desired record and pressing the SELECT key or the MENU key. The LOOKUP window is cleared from the screen and the RETURN field value from the LOOKUP file is "entered" into the editable field on the screen. (If the field is identified in the CAPS statement of the TRS, the RETURN field value will be converted to uppercase before being entered into the editable field on the screen.) If the local RMO field F$UNCKEY is present (see Section 16.15 “F$UNCKEY - Function Key Detection in RMO”), it is set to the value 'LKUP'. Any TRANSFERs specified in the LOOKUP paragraph are also performed.

If no RETURN is specified for a LOOKUP paragraph, TRANSFERs may still be specified if a record is selected in the LOOKUP window. If the local RMO field F$UNCKEY is present (see Section 16.15 “F$UNCKEY - Function Key Detection in RMO”), it is set to the value 'LKUP' and M$M will be set to 'FX'.

If no RETURN is specified for a LOOKUP paragraph CANCEL is the default button for that window.

If no RETURN or TRANSFER is specified in the LOOKUP window the SELECT and MENU keystrokes are ignored.

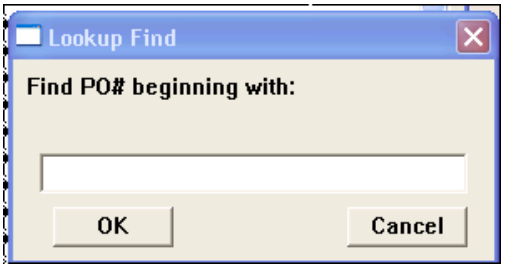
If you press the 

button LOOKUP will prompt for a key value(s).

```
Find <field name> beginning with[42]:
```

42.Find will prompt “**Find <fieldname or descriptor> with value:**” if the field type is numeric or a date

:



where <Field_name> is the [D]KEY1 field's name from the file definition, or a prompt string specified in the PROMPT sub-statement (described previously in this section). If no PROMPT sub-statement is present, LOOKUP prompts only the for the first key field ([D]KEY1). If a PROMPT sub-statement is present, LOOKUP will prompt for each field specified. As each prompt is responded to, LOOKUP will provide values from the current LOOKUP record for any higher key fields whose prompts were omitted in the PROMPT sub-statement, but if a PROMPT statement prompt receives a response of RETURN (a null response), then prompting stops and the target value for all (lower) keys after the last non-RETURN response are set to null. For example, given the following keys and PROMPT sub-statement:

```
Keys:  DEPT X999   KEY1
       ORG  X999   KEY2
       SECT X99    KEY3
       GANG X999   KEY4
       RATE D6    DKEY5
       WEEK I     DKEY6

    PROMPT Department/1 Section/3 Pay_rate/5 Pay_period/6
```

If a response of "801" is given to the prompt:

```
Find Department beginning with:
```

and if a response of "14" is given to the prompt

```
Find Sectiion beginning with:
```

and if a response of RETURN (no value) is given to the prompt

```
Find Pay_rate with value:
```

then prompting stops (Pay_period is not prompted for), and the key search commences using values as follows:

Field Search

Key	Name	Value	Explanation
1	DEPT	801	Response to PROMPT (...Department)
2	ORG	<LOOKUP>	Use Value from current LOOKUP record
3	SECT	14	Response to PROMPT (...Section)
4	GANG	null	Not loaded from current LOOKUP record because RATE reply was RETURN
5	RATE	null	Reply was RETURN
6	WEEK	null	Not prompted because RATE prompt reply was return

Once the key target value(s) are entered, LOOKUP looks for the specified record and if it is found, records are displayed beginning with that record. If the specified key value is not found, records starting with the next previous value are displayed, as in a multi-record screen.

In addition to ***Find,*** Lookup has an optional ***Search*** button. ***Search*** looks for records that contain the target string at any position in the value of the first key (see Section 6.11 "Lookup Windows"). By default ***Search*** is not present. To activate it place the keyword SEARCHBUTTON on a line by itself in the %LOOKUP paragraph.

Alternatively, if you insert the line:

```
lookup.searchbutton
```

in the TRANS_ENV file ***Search*** will be present by default in all lookup windows (and you may use the keyword NOSEARCHBUTTON in the %LOOKUP paragraph to suppress ***Search*** for a specific lookup).

LOOKUP can perform data type conversion on the RETURN field. For example, the field ZIP might be an X99999 in the active file, but might have type A5 in the zip code table. LOOKUP handles such conversions automatically. After conversion, the RETURN value must be a valid value for the field being entered, and must fit within the field's display width. To prevent AdmScreen from giving a field type mismatch error message when data type conversion is necessary, use the /NOTYPE qualifier immediately after the RETURN field name: for example, "RETURN ZIP/NOTYPE".

5.11.1 LOOKUP Window: Examples

The following example specifies a LOOKUP window for field CUST#. When a record from the LOOKUP file is selected, the value from CUSTNO will be entered into field CUST#. In addition, values from fields accessed via the LOOKUP paragraph LINK will be loaded into the screen's IDATE and IAMT fields.

```
E CUST#     %LOOKUP CUSTOMER.IDX
             LINK =I_ INVOICE.MAS KEY IS CUSTNO
             DISPLAY CUSTNAME I_INVDATE I_AMOUNT
             RETURN CUSTNO
             TRANSFER I_INVDATE INTO IDATE
              TRANSFER I_AMOUNT  INTO IAMT
```

The prefix "I_" identifies I_INVDATE and I_AMOUNT as fields INVDATE and AMOUNT from the LOOKUP paragraph LINK to file INVOICE.MAS.

Another example, using the SELECT statement referencing a field in the screen:

```
E WRKORDER %LOOKUP WORKORDER.MAS
            HEADING WORKORDER DESCRIPTION
            DISPLAY W_ORDER/10 DESCR
            RETURN W_ORDER
             SELECT RESP EQ ~EMPL#
```

This LOOKUP paragraph shows only those work orders from the WORKORDER.MAS file where the RESPonsible field is equal to the EMPL# field in the screen virtual record. The ~ (tilde) signals that EMPL# is from the screen rather than the LOOKUP file.

5.11.2 LOOKUP Menu

You can specify multiple LOOKUP paragraphs for a single field. The user then selects a LOOKUP window from a LOOKUP menu that is displayed when the LOOK key is pressed. To specify multiple LOOKUPs for a single field use the %LOOKUP_MENU paragraph as the first entry of a series of %LOOKUP paragraphs.

%LOOKUP_MENU syntax is:

```
E FIELD %LOOKUP_MENU
         [ TITLE Title String ]
         [ WINDOW LINE COL ]
         [ CR_EXIT]
         [ MenuOnCancel ]
         [ IDENT NAME ]
        %LOOKUP ...
```

To utilize a LOOKUP menu specified previously in the current screen, use the name specified in its IDENT statement:

```
E FIELD %LOOKUP_MENU =NAME
```

A maximum of eight LOOKUP entries may be specified for a LOOKUP menu.

Use the %LOOKUP_MENU keyword, %SEPARATOR, indented at least one column in a line by itself, to insert a separator line between entries in the lookup menu.

Note that only LINE and COLUMN numbers are given for the WINDOW statement in a %LOOKUP_MENU paragraph. The compiler will calculate the number of lines and columns needed to display the full menu.

The LOOKUP_MENU window displays the TITLE line from each LOOKUP paragraph as menu items (or the LOOKUP file name if no TITLE is present), preceded by a sequential number. A selection is made either by typing its number or placing the cursor at the desired item and pressing SELECT.

Use the keyword "MenuOnCancel" to cause TRANS to return to the lookup menu if the user clicks on the Cancel button in one of the LOOKUPs chosen from the menu[43].

By defining keystroke macros in the TRANS.ENV file (see Section 6.17 “The TRANS Environment File”) you can enable users to invoke a specific choice from the menu without displaying it.[44] The following TRANS.ENV entries will make F12 automatically invoke menu item 1, and F13 automatically invoke menu item 2:

```
define LKP1=F12 %look 1
```

43. This allows the developer to invoke this feature on individual LOOKUP Menus instead of using the TRANS_ENV "lookup.MenuOnCancel" feature (see Section 6.17.13 “TRANS_ENV Lexicon”) which will force this behavior for all LOOKUP menus

```
define LKP2=F13 %look 2
```

Example:

Assume we have an account structure where some portion of the composite account number sometimes reflects a vendor number, sometimes a customer number, and sometimes an employee number. We may use the following LOOKUP syntax:

```
E ANUMBER %LOOKUP_MENU
           WINDOW 5 10
           TITLE %REVERSE VENDOR/CUSTOMER/EMPLOYEE
          %LOOKUP APP:vendor.mas
           TITLE %REVERSE VENDOR LOOKUP TABLE
           DISPLAY VNO VNAME VCITY VSTATE
           RETURN VNO
          %LOOKUP APP:customer.mas
           TITLE %REVERSE CUSTOMER LOOKUP TABLE
           DISPLAY CNO CNAME CCITY CSTATE
           RETURN CNO
          %LOOKUP APP:employe.mas
           DISPLAY EMPNO EMPNAM EMPCITY
            RETURN EMPNO
```

Pressing LOOK at the ANUMBER field will cause the LOOKUP menu to be displayed (plus a box around it):

```
VENDOR/CUSTOMER/EMPLOYEE
1: VENDOR LOOKUP TABLE
2: CUSTOMER LOOKUP TABLE
3: APP:employee.mas
```

If you want number 2, Customer Lookup Table, either press the '2' key, or press DOWN to make the number '2' choice highlighted, and then press the MENU key.

If the TRANS.ENV file entries described above were implemented, the F13 key would automatically invoke the second entry.

5.11.3 LOOKUP on Local Arrays

LOOKUP windows may be used to access local arrays initialized by the RMO. The syntax for LOOKUP on local arrays is:

```
%LOOKUP_ARRAYS AR1/type(dim) AR2/type(dim) ...
     [ HEADING… | TITLE… | WINDOW…    etc. …]
       DISPLAY AR1 AR2 ...
     [ ARRAY_START Fieldname ]
     [ ARRAY_END   Fieldname ]
     [ RETURN… | TRANSFER… | SELECT… | BOUND…   etc. …]
      ...
```

The arguments that follow the %LOOKUP_ARRAYS keyword, AR1/type(dim), AR2/type(dim) etc., are the names of the local arrays whose elements will be displayed together as "records" in the LOOKUP window. (The field type and size of the local arrays must be declared here, as AdmScreen does not know what is contained in the RMO.)

Note that references to these arrays in LOOKUP sub-statements, e.g. DISPLAY, TRANSFER and SELECT use the base array name only, without any subscript.

The ARRAY_START and ARRAY_END sub-statements are used to identify two integer fields (which may be set by the rmo) that define the range of array elements that are to be displayed (or selected from), e.g.:

44. These macros assume that the order of the %LOOKUP paragraphs do not vary in the .TRS source - you want the application behavior to be consistent.

```
ARRAY_START STARTNO
ARRAY_END ENDNO
```

If STARTNO has a value of 10 and ENDNO has a value of 30 then the %LOOKUP_ARRAYS window would display (or SELECT from) only the 10th through 30th elements of the arrays specified in the DISPLAY sub-statement. If ARRAY_START is not present, the %LOOKUP_ARRAY display will begin with the first element of each array, IF ARRAY_END is not present the dimension of the shortest array named in the DISPLAY sub-statement determines how many elements are used from each array.

The maximum dimension for any array used with %LOOKUP_ARRAYS is 4095.

No KEY or KEY_RANGE statements are allowed with %LOOKUP_ARRAYS.

For example, if the following three arrays are declared in the RMO:

```
XLNAME/A20(20) Davis Neer Grahl Saether Piecham Yee Saether
XFNAME/A20(20) Bill Avi Bart Dagfinn Chuck Ginny Kjell
XAMT/D2(20)    4.29 6.02 0 17.23 18,429.67 0 23
```

Then the following %LOOKUP_ARRAYS paragraph could be used to access the arrays in a LOOKUP window:

```
E D2 %LOOKUP_ARRAYS XLNAME/A20(20) XFNAME/A20(20) XAMT/D2(20)
     SELECT XAMT GT 0
     DISPLAY XLNAME XFNAME XAMT
     RETURN XAMT
```

The above code would result in a LOOKUP window containing the following:

XLNAME	XFNAME	XAMT
DAVIS	BILL	4.29
NEER	AVI	6.02
PIECHAM	CHUCK	18,429.67
SAETHER	DAGFINN	17.23
SAETHER	KJELL	23.00

5.12 Menu Bar

TRANS' Menu Bar allows the screen developer to present the user with a selection of options utilizing the standard Windows Menu Bar.

The choices presented in a menu bar can be any type of TRANS function: perform a branch, do a calculation, file a record, move the cursor, spawn a process, leave TRANS, etc. A menu bar choice can also display a submenu.

In TRANS, the Menu Bar is activated by pressing the MENU key when positioned at the first character of any field. Section 6.12 “Menu Bars and Submenus” describes how to navigate once the Menu Bar is active.

Each choice on the menu can either

1. cause a branch,
2. cause a special RMO call where S$S is the field name you were at when you entered the menu and M$M is a value specified in the BAR paragraph,

3. leave TRANS,
4. call TRANS HELP,
5. display the TRANS branch menu; or
6. display a submenu with further choices of the first 5 kinds.

5.12.1 Bar Paragraph Syntax

BAR syntax in the TRS is:

```
BAR     Choice_name Action_code  [Parameters]
    Description
    ...
```

The BAR statement is placed above the SCREEN statement, and there can be only one BAR per screen. Each item in the bar is described by 2 lines; and the item description lines must be indented with at least one space or tab.

After the BAR statement up to 20 pairs of lines can be used to specify up to 20 menu choices.

```
...
Choice_name Action_code  [Parameters]
Description
...
```

The first line of each pair identifies the menu item (Choice_name), defines the type of action this choice is to perform (Action_code), and supplies any parameters that type of action might require. The second line contains a short description for the item, up to 40 characters long (blanks may be included.)

Choice_name is a short string of up to 8 characters with no embedded blanks, which will be displayed in the bar menu.

Action_code must be one of the values listed in the following table:

Action Code	Description	Parameters
QUIT	leave TRANS	none
BRANCH	branch	branch code or *
SUBSCREEN	subscreen	subscreen name or *
EXECUTE	calls RMO	M$M value
HELP	calls HELP	help topic name
MENU	calls submenu	submenu name

BRANCH requires a branch code as a parameter. If a menu choice is selected that has the BRANCH action code, TRANS will branch to the target screen indicated by the supplied branch code. If the branch code parameter is an asterisk (BRANCH *) rather than a branch code, TRANS will call up the standard branch menu when the item is selected. By using BRANCH *, you can make all manual branches available as one menu item, instead of having to specify each branch as a separate menu item.

Similarly, the SUBSCREEN action code can be used to display a particular subscreen by giving its name as the parameter, or to display the subscreen menu, by giving an asterisk as the parameter.

EXECUTE requires a value of M$M as a parameter. If a menu choice is selected that has the EXECUTE action code, TRANS will call RMO with the mode, M$M, set to the value supplied in the EXECUTE parameter, and the status, S$S,, set to the name of the field from which the menu was called (i.e. the field at the current cursor position).

MENU requires the name of a submenu as a parameter. See Section 5.12.2 "The Menu Paragraph" for a discussion of the MENU paragraph.

HELP requires the name of a help topic in the ADM$HELPFILE. If a menu choice is selected that has the HELP action code, TRANS will call HELP for the topic name supplied by the HELP parameter. If the HELP parameter is H$ELPNAME,[45] HELP will use the contents of the field H$ELPNAME to identify the topic section to look for.

5.12.2 The Menu Paragraph

If the BAR paragraph offers any choices with the MENU action code, then each submenu named must be specified in a MENU paragraph. All MENU paragraphs must be placed below the BAR paragraph in the TRS.

MENU paragraph syntax in the TRS is similar to BAR paragraph syntax (see Section 5.12.1 "Bar Paragraph Syntax"):

```
MENU Menu_name
   [SEPARATOR]
   Choice_code  Action_code  [Parameters]     Description
   [SEPARATOR]
   [Choice_code etc. …]
```

The MENU line must contain a menu name: This is the name by which the submenu is referenced in the BAR paragraph.

The menu items are described in the same way as in the BAR paragraph. Choice codes should only be one character long, and should be unique within each MENU paragraph, to enable accelerator keys.

A separator (horizontal graphic line) can be inserted in a submenu with the keyword "SEPARATOR" ("se" is enough) on a line by itself in the MENU paragraph. Normally a separator is placed between two items, but it can also be placed above the first item, to draw a line under the submenu title area.

Example:

```
MENU Verify
   N EXECUTE YN
   Change Name
   A EXECUTE YA
   Change Address
   SEPARATOR
   X QUIT
    Accept and Exit
```

There can be any number of SEPARATORs in a submenu, but they do count against the limit of 20 items in a menu paragraph. Item descriptions are limited to 40 characters.

Only one level of submenus is supported. Therefore, the action code 'MENU' cannot be used in a MENU paragraph.

45. See Section 6.14 "HELP in TRANS"

5.12.3 Enhanced Accelerator Capability

If the tilde character (~) is placed in the string assigned to the logical name OPTION, an enhanced accelerator key capability is enabled. This option makes accelerators in the menu bar and submenus more flexible, makes TRANS display them differently, and causes AdmScreen to error check them. This is a SCREEN (i.e. compile-time) option; to get the enhanced accelerators you must re-SCREEN the TRS with "~" in OPTION.[46]

With enhanced accelerator keys enabled, TRANS displays the accelerator key for a menu bar or submenu item in underlined boldface, and **any character in the menu bar item name may be the accelerator key for that item**.

The Menu BAR item syntax is:

```
Name Action [Parameter]
Description
```

"Name" is what is displayed in the bar. The default accelerator for the item is the first character of "Name" (case blind). If the screen is compiled with "~" in OPTION the first character will be highlighted with underlined boldface, to identify it as the accelerator key, unless another character in the item name is designated as accelerator for that item by placing a tilde (~) before it: e.g., "E~xit" tells AdmScreen to highlight and use 'x' (and 'X') as the accelerator key for item "Exit".

If more than one tilde appears in a name, only the first one is significant; any other tildes are just literal characters. If a tilde appears at the end of a name ("Name~"), it is a literal character. These rules apply to MENU item syntax as well.

The MENU item syntax is:

```
C Action [Parameter]
Description
```

46. It does not matter whether OPTION "~" is assigned when TRANS runs, once compiled the enhanced behavior is "built-in" to the screen.

By default, the choice code and the description for each item appear in the submenu. If the screen is compiled with "~" in OPTION only the description is displayed in the submenu, with the accelerator key highlighted in underlined boldface. By default, the accelerator key is the choice code (AdmScreen will check that the choice code character is present in the description, issuing a warning message if it is not[47]).

When the "~" is used to designate an accelerator, the choice code is not used, but is still required as a placeholder.

5.13 The MESSAGE Facility

The MESSAGE facility allows screens to display messages whose contents are determined at run-time, whenever the cursor arrives at a field. The message disappears when the cursor leaves the field. Anything hidden by the message is refreshed.

The MESSAGE paragraph is an optional component of a field declaration, identified by the keyword %**MESSAGE.** The syntax for the MESSAGE paragraph follows.

```
E FIELD %MESSAGE [LOCATION][ALIGN][BOX][VIDEO][EXECUTE][=NAME]
[IDENT msg_nm]
[CONTROL field_name]
DISPLAY literal ^ --FIELD FIELD-- $$FIELD FIELD.--
[DISPLAY additional lines]
```

The only required items are the %MESSAGE keyword itself and at least one DISPLAY substatement (or a reference to a previously described message statement.) "!" for comments can be used everywhere except after DISPLAY, which interprets '!' as a literal character in the message text.

47. When "~" is in OPTION, AdmScreen also checks that the accelerators are unique for a given submenu or menu bar.

The keywords described below are all optional. If used, they **must** appear on the same line as "%MESSAGE".

LOCATION	There are several ways to specify where to display the message.
Coordinates	(4 numbers) the position and size (line, column, height, width) of the window where the message is displayed. The line and column numbers are relative to the upper left corner of the physical screen. Because using coordinates and dimensions can be tedious and hard to maintain; there are four keywords you can use instead: ABOVE, BELOW, RIGHT, and LEFT. With these keywords, the width and height of the message window are determined by whatever it displays; and the screen placement is determined using the coordinates of the left end of the field.
ABOVE	Places the message above the field.
BELOW	Places the message below the field.
RIGHT	Places the message to the right of the field.
LEFT	Places the message to the left of the field. If none of these keywords is used and no coordinates are given, the default message window is assumed. It starts in column 1, two lines above the bottom line on the screen; it is two lines high and uses has a width ten characters less than the entire screen width, e.g. on a 24x80 screen, the default message window starts on line 22, column 1 and extends to line 23, column 70.
ALIGN	Two mutually exclusive keywords modify the alignment of the message window.
CENTER	If present, modifies the alignment of the message window when ABOVE or BELOW is specified, so that the message window is centered above or below the cursor when it arrives at the field. If CENTER is not specified, the left side of the message window is aligned with cursor when it arrives at the field.
RJUST	If present, right-justifies the message within the message window. Used with ABOVE or BELOW RJUST also aligns the right side of the message window with the right end of the field.
BOX	If BOX is present, the message window is displayed inside a box. If coordinates are used to locate and size the message window, the height and width must include two lines and two columns for the box.
VIDEO	One of more video attributes can be specified for the message window, using the following keywords.
BOLD	Displays **both** the text of the message and the surrounding box (if any) in bold.
UL	Underlines the full width of the message window. UL has no effect on the box.
REVERSE	Displays the full width of the message window in reverse video. REVERSE has no effect on the box.

EXECUTE	Tells TRANS to call the RMO just before the message is displayed. The RMO can set fields in DISPLAY statements or can set a CONTROL field (described below). The RMO is called with S$S set to the field name and M$M set to 'MS'. If the screen has no RMO, EXECUTE is ignored.
=NAME	Used to reference a previously declared message by its IDENT name (described below). When a previously declared message is referenced, the %MESSAGE line is the only line of the message paragraph, i.e. the IDENT, CONTROL, and DISPLAY lines cannot be overridden. If the referencing message has nothing except "=NAME" on the %MESSAGE line, all %MESSAGE line options are copied from the referenced message. Otherwise, the referencing message uses the options specified on its %MESSAGE line. The "%MESSAGE" keyword appears on the same line as the field declaration, after the precise placement specification (if any), unless the field has any other "%" keywords ("%WINDOW", "%LOOKUP", %BOLD", etc.). If any other "%" keywords are in use for the field, %MESSAGE (indented) goes on the next line following the field declaration. %LOOKUP (see Section 5.11 "LOOKUP Window") can be used with %MESSAGE in either order. Just start the %MESSAGE paragraph (on a new line, indented) immediately after the last line of the %LOOKUP paragraph, or vice-versa. The MESSAGE paragraph sub-statements described below, if used, are placed one to a line on indented lines following the %MESSAGE line. Unless the MESSAGE paragraph is referencing a previously specified MESSAGE (using the =NAME syntax), the message paragraph must contain at least one DISPLAY statement.
IDENT	Gives the message a name so it can be re-used for another field via "=NAME". Only the first four characters of the IDENT name are used by TRANS. These first four characters must be unique within a screen.
CONTROL	Allows the RMO running with the screen to determine whether or not to display the message. Identifies an integer (I) field which TRANS is to check. If the CONTROL field is set to zero the message is not displayed; or if it is set to one the message is displayed. CONTROL can be used with or without EXECUTE, and it must be above the first DISPLAY statement.

DISPLAY	Provides a layout for one line of the message. At least one DISPLAY statement is required, and there is no limit on the number of DISPLAY lines except that they must fit on the screen. Each DISPLAY contains a line of field designators and/or literal text to be displayed. Field designators in DISPLAY must have one or more dashes (-) or dollar signs ($) in front of the full or partial field name (e.g.: ---FLD, $$$FLD), or else one or more dashes after the field name (e.g. FLD---). As in AdmScreen or REPORT layouts, the total width of the field name string with its dashes or dollar signs is the maximum width which will be displayed (values of the field which are too wide are truncated on the right). Leading dollar signs cause the '$' character to be displayed immediately before the value. To force a match on the exact field name string given, put a dot after the field name (e.g., '---FLD.' or 'FLD.---'). Displayed numeric values always have leading zeroes and commas suppressed. Hats (^) can be used to insert "hard blanks" in DISPLAY literals: this is especially useful if you want a RIGHT or LEFT message to start or end with blanks in order to separate it from the field's value. DISPLAY followed by nothing generates a blank line. The message lines are formed by concatenating the various literals and fields on each DISPLAY line, with one blank between them (unless hats are used to insert "hard" blanks). It makes no difference whether dashes are on the left or the right of a field name, because leading and trailing blanks in field values are ignored.
TABULAR	Allows you to create a tabular display in a %MESSAGE paragraph, with properly justified and aligned columns. TABULAR can be placed on the %MESSAGE line and affects the formatting of the message display in two ways: 1. Multiple blanks between fields or between literals and fields in message DISPLAY lines are preserved, not squeezed out. 2. As in REPORT, fields in DISPLAY are left justified or right justified, depending on which side of the field name the dashes or dollar signs are placed. The TABULAR attribute is inherited by “=name” references to a message. That is, if a message has TABULAR, other messages which reference it by its IDENT name have the TABULAR attribute. If a message does not have TABULAR, messages referencing it do not, either. The TABULAR keyword is not required, and is ignored, on %message lines which contain an “=name” reference. When TABULAR is used, hats (^) are not needed to preserve blank space.

The message is displayed after fields are displayed. The developer must therefore make sure that a message using explicit coordinates and dimensions does not cover up the display of the field to which it refers.

A MESSAGE window and a LOOKUP window on the same field can overlap on the screen. When the LOOKUP window displays it may cover part of the message window; when the LOOKUP window clears the MESSAGE window is refreshed on the screen.

5.13.1 MESSAGE Facility Example

VEH.TRS and VEH.RMS, listed below, specify a screen that illustrates some uses of %MESSAGE. Note that field JCDE has both a video keyword (%BOLD), a LOOKUP paragraph, and a MESSAGE paragraph; and that the display of the message for field VID is controlled via logic in the RMS.

```
***** VEH.TRS *****
*
VEH DATA$DIR:VEH.MAS 1 VEH.RMO
ER AUTHCODE/A10
DR AUTHFLAG/I
E JCDE %BOLD %LOOKUP JCDE.MAS
      DISPLAY JCDE JDESC
  %MESSAGE BELOW CENTER BOX
      DISPLAY To update a different job code than
      DISPLAY JCDE----- type the new value
      DISPLAY over the current value
      DISPLAY then press RETURN
*
E VID %MESSAGE BELOW CENTER BOX BLINK EXECUTE
      CONTROL AUTHFLAG
      DISPLAY You must enter
      DISPLAY an authorization code
      DISPLAY for changes to be valid!
*
SCREEN
             Vehicle Assignments
BL
      Authorization Code: AUTHCODE-------
BL
         Job Code              Vehicle ID
       JCDE-----               ------VID
END

***** VEH.RMS *****
*
FILE DATA$DIR:VEH.MAS
M$M/A2
S$S/A6
AUTHFLAG/I
AUTHCODE/A10
PROGRAM
IF M$M EQ 'MS' AND AUTHCODE EQ ' ' THEN ; !Display when
    AUTHFLAG = 1 ELSE AUTHFLAG = 0 END   !auth code nonblank
```

5.14 %HOVER - Hover Text for fields

The optional %Hover paragraph specifies "hover text" messages that "pop up" when the mouse cursor passes over the field. The syntax is similar to the %MESSAGE paragraph described above:

E FIELD %HOVER [LOCATION][ALIGN][BOX][VIDEO][EXECUTE][=NAME]
[IDENT msg_nm]
[CONTROL field_name]
DISPLAY literal ^ --FIELD FIELD-- $$FIELD FIELD.--
[DISPLAY additional lines]

5.15 Subscreens

The SUBSCREEN facility provides a means to display a subset of a screen's fields, literals, etc. Subscreens can be laid out side by side, or they can completely or partially overlap.

Subscreens are useful, for example, in data entry screens where there are too many fields to fit legibly on the physical screen, or in screens where different subsets of fields are displayed and edited under different conditions.

Using subscreens can reduce the number of screens and RMOs and the amount of duplicative code in the application, which makes both implementation and maintenance easier. In addition, the use of subscreens can improve the responsiveness of the application to the extent that branching to complex screens is reduced. Switching between subscreens is much faster than branching because all data files are already open and the TRO and the RMO are already in memory.

Note: Subscreens are not available in multi-record screens, but a subscreen (of a single-record screen) may be a multi-record subscreen.

Subscreens are only a mechanism for changing the screen display and the list of fields on the screen. A subscreen is not a separate screen: instead, a main screen and all of its subscreens make up a single TRANS screen. The RMO and all LINKs, etc., operate exactly as if there were no subscreens. All TRANS limits and syntax rules apply to the entire screen, not to each subscreen separately.

A screen can consist of a main screen and up to 14 subscreens. TRANS initially displays only the main screen. In the main screen, the only editable or refreshable fields are those in the main screen itself. The user or the RMO can then invoke various subscreens. When TRANS displays a new subscreen, the cursor normally lands on the first editable field in the subscreen. However, in a subscreen, both the main screen's fields and the fields in the subscreen are editable and refreshable: with the usual keystrokes (HOME, arrows, RETURN) the user can place the cursor at any of these fields. All TRANS features and keystrokes are available within subscreens. Specifically, since the key fields should be in the main screen, the user can enter a key value to go to another record without changing subscreens.

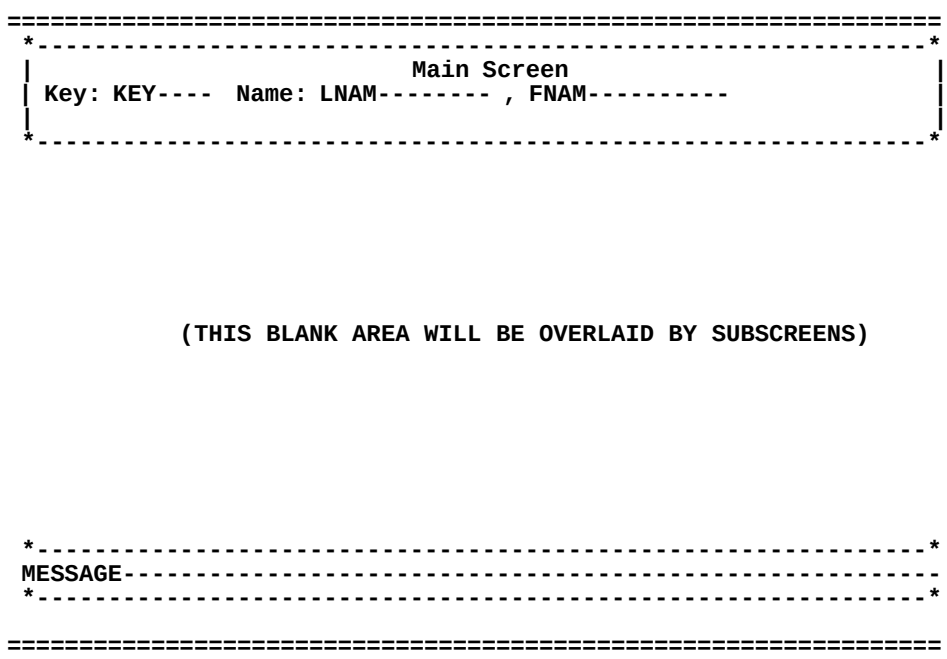
The SUBS keystroke is used to move between subscreens. When SUBS is pressed, the subscreen menu appears. This menu looks and works like the branch menu (see Section 5.7.3 “Calculated Branches”). The main screen always has "branch code" 0, while subscreens are numbered from 1 to 9 and A to E. Subscreens appear in the menu in the same order they appear in the TRS. Using the menu, the user can go to any subscreen in the current screen, or to the main screen.

5.15.1 Subscreen Design Considerations

Fields on the main screen are **always** refreshed, and so when switching from a subscreen to the main screen, the main screen is **not** redisplayed **(the main screen is always assumed to be visible!).** Because of this, the following guidelines should be observed when designing screens with subscreens.[48]

1. All key fields used in the screen should appear in the main screen.
2. Any other fields which should always be visible, editable, and/or visibly refreshed should be on the main screen.
3. Other fields, especially other editable fields, should appear in subscreens, not in the main screen.
4. Subscreens should not overlap with any non-blank part of the main screen. The net result is that, generally, the main screen will have very little on it.

For example, a main screen might look like this:



In the above example, the key and name fields, and the message field, are always visible, editable, and refreshable. Various subscreens with additional fields, literals, boxes, etc., would be laid out in the blank area as described below.

48. Some special notes on Digital's VTxxx terminals: VTxxx compatible terminals cannot display part of the screen in 80 column mode and another part in 132 column mode. Therefore, all subscreens have the same character mode (80 or 132) as their main screen. Double width (DW) and double height (DH) are attributes of an entire line on a VTxxx screen, not attributes of a character cell. These features should be avoided in screen layouts which do not use the full width of the screen.

5.15.2 Subscreen Syntax

Subscreens are implemented using three statements: SUBSCREEN, TITLE, and SCRMENU. These keywords fit in the TRS syntax as follows:

```
                Screen header line
                [ LINK, APPEND, and INDEX paragraphs ]
-->             [ TITLE Main_screen_title ]
-->             [ SCRMENU L# C# HGT WID Subscreen_Menu_Title ]
                Main Fields section
                SCREEN [ coordinates ]
                Main screen layout
                *
-->             SUBSCREEN Subscreen_name [RPS[/n] [RESTORE] [LOCK]
                [ TITLE Subscreen_title ]
                Subscreen Fields section
                SCREEN [ coordinates ]
                Subscreen layout
                *
                SUBSCREEN Subscreen_name ...
                ...
                *
                [ BRANCHES ]
                END
```

SUBSCREEN is **required** if subscreens are used, the others are optional.

The TITLE statement provides a screen description (up to 40 characters long) which appears in the subscreen menu. TITLE should be present in the main screen and in each subscreen. If there is no TITLE for the main screen or a subscreen, the subscreen menu displays "MAIN" or the subscreen name. To prevent a specific subscreen (or main screen) from appearing in the subscreen menu, use the statement 'TITLE %%'.

The SCRMENU statement gives the developer more control over the subscreen menu. The coordinates and dimensions of the subscreen menu box can be specified, and a subscreen menu title can be given. The SCRMENU syntax is the same as the BRANCHES syntax for customizing the branch menu (see Section 5.7.1 “Customizing the "Pop-up" Branch Menu”), except that the menu title is limited to 40 characters. If used, SCRMENU should appear only once per screen, in the main screen section. If SCRMENU is not present, the subscreen menu appears centered near the bottom of the screen with the title "Subscreen Menu".

If TITLE or SCRMENU is used in the main screen, they must appear below all external file paragraphs (LINK, APPEND, INDEX).

SUBSCREEN paragraphs are described between the last line of the main screen layout and the BRANCHES or END statement. The SUBSCREEN header line indicates the beginning of a subscreen and gives it a name (subscreen names must be unique within each screen and are limited to 18 characters in length). Each screen can have up to 14 subscreens.

RPS is an optional number to indicate the number of records in a **multi-record subscreen**. For example,

```
subscreen listall 5
```

Specifies the subscreen will display 5 records at a time.

If each record in the multi-record subscreen is to have multiple lines, specify the number of lines for each repeating record by appending a slash (“/”) and the number to the RPS number. For example,

```
subscreen listall 4/3
```

Specifies the subscreen will display 4 records at a time with three lines being displayed for each record.

The **RESTORE** keyword on the SUBSCREEN line causes the subscreen to disappear when the user goes to another subscreen. Whatever was covered up by the subscreen is redisplayed.

The **LOCK** keyword on the SUBSCREEN line prevents the user from going to another record while in the subscreen. Using LOCK, the key fields become display-only; and all record movement keystrokes are blocked (SELECT, PREV, NEXT, NREC, etc.).

Each SUBSCREEN paragraph has its own fields section. The subscreen fields section has the same purpose, syntax, and supports all the features of the main screen fields section, as described in Section 5.5 "Field Names", i.e.: virtual fields, check and message statements, CAPS, REQUIRE, LOOKUP, precise placement, BOX, etc. As in any other screen, BOX coordinates and precisely placed field coordinates are relative to the upper left corner of the SCREEN rectangle, which is specified in the SCREEN statement of the SUBSCREEN paragraph.

Both the VIDEO statement and special BOX statements such as BOX DEFAULT carry over into any subsequent SUBSCREEN paragraphs in the screen: they need not be repeated in each subscreen's fields section.

LINK, APPEND, and INDEX paragraphs as well as BAR and MENU statements must appear in the main screen, not in subscreens.

Each subscreen also has its own SCREEN layout section. Normally, the SCREEN statement for a SUBSCREEN will include coordinates and dimensions for the subscreen rectangle (a "split screen"), so that it will not overlap with information displayed in the main screen.

5.16 Text Fields

Text fields (TInn and TXnn data types) are edited and displayed using the AdmTed editor, as described in Appendix J.1 "AdmTed".

Until AdmTed is called (via the EDIT keystroke) only the first line of the text field is displayed by default, following the same display syntax that is used for alpha fields, e.g. to display field EXPLANATION, data type TI60, use:

```
E EXPLANATION [2,59,21]  ! in the field declaration section.
                      or
EXPLANATION----------    ! in the SCREEN description
```

In either case only the first line of EXPLANATION will appear on the screen, truncated to 21 characters, until AdmTed is called for EXPLANATION.

To display multiple lines of the text field use the

```
TEXTFIELD----n------
```

syntax, where ***n*** is the number of lines in the text window. AdmTrans will create a (display only) Rich Text edit box and load it with the text from the field, and if necessary supply vertical scroll bar (the text will be reformatted to fit the horizontal size of the edit box). The multi-line Rich Text edit box can also be specified using precise placement syntax:

```
E TEXTFIELD [line,col,#lines,#columns]
```

If the internal text editor is AdmTed (only!) AdmTrans supports use of the %WINDOW syntax to open AdmTed in a specified area. E.g.

```
E TEXTFIELD [10,30,15,50] %WINDOW 10 30 15 50
```

will create an edit box for display beginning line 10, column 30, 15 lines high and 50 columns wide, and overlay the exact same area with an AdmTed window when the edit key is pressed.

The reserved field TED$TITLE/A16 (or larger) can be used in the virtual record to specify an alternative to the default window title for the AdmTed session that launches from AdmTrans to display or edit a text field. For example:

```
TED$TITLE/A30 'Edit Delivery Instructions'
```

The additional text flag

```
"**** Read Only ****"
```

is always appended to the window title if the file is opened read only.

If the text field is declared as a display-only field in the field declaration section, or if the field is in fact display-only because, for example, the file has been opened read-only, or because the EDFLDS subroutine made it display-only, the AdmTed editor is called in read-only mode, i.e. the file can be viewed but not altered.

If the text field is editable the AdmTed editor is called with full editing capability. The file may be initialized, either directly via the Data Dictionary, as described in Appendix J.8 "The Text Initialization File", or under the RMO control, as described in Section 16.24 "TX$INITF: Automatic Initialization of Text Fields".

5.17 Parameterization

If the letter "p" (lowercase) is included in the string assigned to the logical name OPTION (see Appendix A: "Options"), then any string in the TRS instruction file can be parameterized by placing the string in angle brackets. The string enclosed by the angle brackets becomes a prompt that will be displayed during the compilation of the screen.[49] The string typed in response to the prompt by the user is inserted into the instruction file **for the purpose of the particular compilation** in place of the angle bracketed parameter text. For example:

```
BILLING <Enter year to view>BILL.MAS 1 BILLING.RMO NOMSG

V DEPARTMENT/A20 <Enter department name>

CE Job status for <Enter your name>
```

Once text has been supplied for a particular parameter, i.e. a particular angle bracketed string, then that text will be substituted for the parameter each time it is encountered.

If the parameter is enclosed in double brackets, as follows:

```
ER NAME/A40 [5,<<Enter column for name, or press return to
skip>>,40]
```

and the user does not supply a response, then AdmScreen will ignore the entire instruction line which contained the double bracketed string.

5.17.1 Logical Parameters

If the parameter string contained in the angle brackets begins with the characters "L_", (e.g. <L_fieldname>), then AdmScreen first tries to translate the prompt as a logical name. If the logical name has been assigned the user is not prompted for the contents of the parameter. Instead the value of the logical name is substituted for the prompt. Parameters which begin with the characters "L_" and are assigned as logical names are called "logical parameters".

When the logical names exist, the display of logical parameter prompts and their values can be suppressed by assigning the lowercase letter "c" to the logical name OPTION (see Appendix A: "Options").

If a parameter beginning with "L_" is not assigned as a logical name, then the user is prompted for a value as in standard parameterization (see Section 5.17 "Parameterization").

Prompting for values when the logical name is not assigned can be avoided entirely by supplying a **default value** in the parameter string, as follows:

```
<L_MINIMUM=0>
```

Specify the default value for the logical name by appending ***"=value"*** to the logical name inside the angle brackets. In the example above if the logical name L_MINIMUM is not assigned, the value "0" will be substituted for the parameter.

5.17.2 Data DIctionary Parameters

If the parameter string contained in the angle brackets begins with the characters "L%", "W%", or "J%" (e.g. <L%fieldname>), where "fieldname" is a field in the virtual record that references an element in the Data Dictionary, then AdmScreen first tries to substitute the prompt with the "Line Label" (L%), "Display Width" (W%) or "Justification" (J%) attributes of the referenced Data Dictionary element. as a logical name. If successful AdmScreen does not prompt the user for a value.

As with logical parameters, when Data Dictionary references exist, the display of parameter prompts and their values can be suppressed by assigning the lowercase letter "c" to the logical name OPTION (see Appendix A: "Options").

---

49. Because AdmScreen can accept wildcard syntax (*.trs) to screen many TRSs in one step, parameterization should be used carefully with AdmScreen. We recommend using logical ("L_") parameters (see Section 5.17.1 "Logical Parameters") in TRSs, and never using the same logical name for parameters which have different values in different TRSs. If you do not use logical parameters, AdmScreen prompts for parameter values; with a wildcarded AdmScreen command, it prompts once for each unique parameter string within each TRS. Consistent use of logical parameters eliminates the prompting and makes AdmScreen wildcard syntax much easier to use.

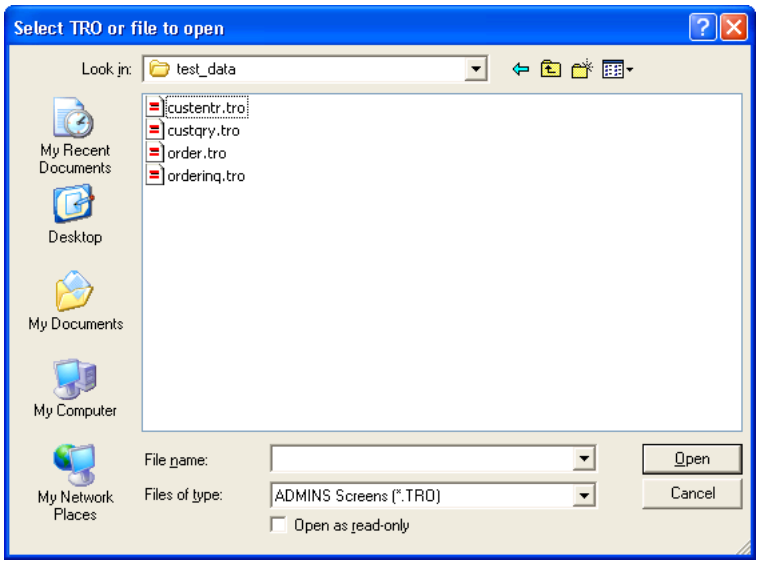
If a parameter beginning with "L%" does not succesfully retrieve a value from the Data Dictionary, then the user is prompted for a value as in standard parameterization (see Section 5.17 “Parameterization”).

Chapter 6:TRANS: Screen Transactions

AdmTRANS (or “TRANS”), the TRANSaction processor, is used to process data in a screen display. Simple screen formats can be generated automatically by TRANS "General Editor Mode", which is described in Section 6.16 “General Editor Mode (“GENED”)”. Generally, though, specific screen instruction files are first "compiled" using the AdmScreen (or “SCREEN”) command, described in Chapter 5: “AdmScreen: Compiling Screen Forms”. The SCREEN command produces a file called "name.TRO" by compiling the screen instruction file called "name.TRS". Then the user can type "TRANS name" on the command line to call TRANS to process data according to that screen. For example:

```
$ trans name
```

If the user types only TRANS on the command line, TRANS will prompt for a screen-name or a file name:



Clicking “Cancel” will exit TRANS. The FILE-NAME option (General Editor mode) is described in Section 6.16 “General Editor Mode (“GENED”)”.

On Windows, AdmTrans is usually designated as the default executable for file type “.TRO” - so that a TRANS session for a particular TRO can be started simply by clicking on the TRO (or a shortcut for the TRO) on the desktop or in a folder.

6.1 Command Line Options

AdmTrans supports two command line options that provide control of the logical name environment for the AdmTrans session, and any child processes of the AdmTrans session.

6.1.1 /Job command line option

The command line option "***/job=n***" where n is any 1-5 digit number tells AdmTrans to use an alternative process logical name table instead of the default table (Job00001).

For example, the command:

```
AdmTrans /job=7 OBJ:Myscreen
```

tells AdmTrans to use the Job00007 process logical name table instead of the default Job00001 table (or whichever table the environment variable ADM_LOGICAL points to)[1].

Alternatively, /job can be specified as

```
/job=%pid%
```

to utilize the process id of theAdmTrans session as a unique process logical name table identifier.

AdmTrans issues a statement similar to

```
set ADM_LOGICAL=7
```

so that any process initiated from AdmTrans (e.g. spawned reports) will use the Job00007 process logical name table.

Assume you have a major application that uses the default Job00001 table, and that you want to create a copy of the application for training, testing or further development. The two versions of the application use the same logical names, so the test/training copy should be set up to use a different process logical name table, e.g. Job00007.

You could launch a Command Window and execute this command (automatically or manually):

```
set ADM_LOGICAL=7
```

then all commands (including AdmTrans) executed from that Window will use the job00007 table.

But lets say you launch the production application via a shortcut to start AdmTrans (perhaps using a startup "home" or menu screen) where the Target variable for the shortcut is:

```
C:\ADMINS\bin\AdmTrans MYAPP:MyScreen
```

Create a copy of the shortcut and change the Target to:

```
C:\ADMINS\bin\AdmTrans /job=7 MYAPP:MyScreen
```

and you will have a shortcut to the test/training version of the application (AdmTrans will change the ADM_LOGICAL environment variable and use the Job00007 table - for itself and all its children).

---

1. See Appendix C.4.5 "Managing Process Logical Names"

6.1.2 /Lnm command line option

The command line option "***/lnm=file_name***", where file_name is the path of a text file that contains logical names and values (the same file structure that is described in Appendix C.4.1.1 "Create multiple logical names using a file") tells AdmTrans to create the specified logical names at startup before launching the screen.

If you intend to create Process logical names, and you use the

```
/job=n
```

command line switch to force AdmTrans to use an alternative Process Logical Name table (as described above in Appendix 6.1.1 "/Job command line option") ***"/job=..."*** **must appear before *"/lnm=..."* in the command line for the process logical names to end up in the right table**.

6.2 Standard Functional Keystrokes

The table that follows shows the default mapping of TRANS' standard function keystrokes to the physical keyboard, as well as all the physical keystrokes that are recognized by TRANS. .

Physical Name[a]	Key Name	TRANS Function Key	Description	F$UNCKEY=physical (if TRANS function key)
CT_K	Ctrl/K	altx	Pop Alt. Index Menu	CK
CT_E	Ctrl/E	apnd	Toggle Append and Update mode	CE
F9	F9	brnc	Pop branch menu	
CT_C	Ctrl/C	copy	Copy value from previous record. Edit Mode: Copy field value to the clipboard.	CC
DELE	Delete	del	Delete record.	
DOWN	Down-arrow	down	Cursor down	
F4	F4	edit	Enter edit mode[b] (for text field)	
CT_B	Ctrl/B	exit	Exit from TRANS	CB
CT_W	Ctrl/W	fatt	Pops Field Attribute dialog box	CW
CT_Q	Ctrl/Q	fsm	Field select toggle (Query/Tab mode)	CQ
F1	F1	help	User level help (see Section 6.14 "HELP in TRANS")	

Physical Name[a]	Key Name	TRANS Function Key	Description	F$UNCKEY=physical (if TRANS function key)
HOME	Home	home	Move the caret to the first key field. Edit Mode: Cursor to beginning of line	
CT_G	Ctrl/G	ins	Insert another record with this key	CG
LEFT	Left-arrow	left	Cursor left one field Edit Mode: Cursor left one character	
CT_D	Ctrl/D	lkcl	Close display-only LOOKUP	CD
CT_U	Ctrl/U	lkdo	Launch LOOKUP in display only mode	CU
F5	F5	lksf	Pop Lookup for sub-field	
F3	F3	look	Pops Lookup Window. (See Section 6.11 "Lookup Windows")	
ALT	Alt	menu	Brings focus to the first menu item	
CT_F	Ctrl/F	msg	Toggle field format display	CF
F12	F12	nbrk	Next break (multi-record)	
PGDN	Page Down	next	Next record/screen-full of records	
CT_N	Ctrl/N	nrec	Move up/down N records (pops dialog box)	CN
F11	F11	pbrk	Previous break (multi-record)	
CT_A	Ctrl/A	plus	Adds one if a numeric field	CA
PGUP	Page Up	prev	Previous record/screen-full of records	
CT_P	Ctrl/P	prt	Put an image of the screen on the clipboard and (optionally) print it.	CP
RGHT	Right-arrow	rght	Cursor right Edit Mode: Cursor right one character	
CT_L	Ctrl/L	rmo	Call the RMO with status (S$S) set to the field the cursor is on, and mode (M$M) set to 'XX'. Allows special RMO processing regardless of cursor position (See Section 16.18 "Special Keystroke to Call the RMO" for details).	CL

Physical Name[a]	Key Name	TRANS Function Key	Description	F$UNCKEY=physical (if TRANS function key)
F2	F2	shfk	Pops the Show Function Keys dialog	
F6	F6	subs	Pop sub-screen menu (see Section 6.9.1 "Subscreens")	
CT_O	Ctrl/O	tdbg	Toggle Debug Mode (not implemented)	CO
CT_Y	Ctrl/Y	tint	Bring up TRANS internal debug dialog boxes.	CY
CT_T	Ctrl/T	trf	Transfer this record to a new key.	CT
UP	Up-arrow	up	Cursor up	
F8	F8	xfwd	Forward to screen before last xret	
F7	F7	xret	Return to previous screen	
BACK	Backspace		Edit Mode: Erase character	
TAB	Tab		Cannot be used as a TRANS function key. Moves the caret from field to field	
F10	F10		Acts like ALT (unless F10=appkey in TRANS_ENV)	
END	End		Edit Mode: Move the cursor to the end of the line	
INS	Insert		Edit Mode: Toggle insert/ overwrite mode	
CT_H	Ctrl/H			
TAB	Ctrl/I		Same as the TAB key	
CT_J	Ctrl/J			
RET	Enter (Ctrl/M)		Cannot be used as a TRANS function key. Only legal use is in SETKEY strings to signal a C.R., or as a F$UNCKEY setting.	
CT_S	Ctrl/S			
CT_V	Ctrl/V		Edit Mode: Paste the clipboard into the field	
CT_X	Ctrl/X		Edit Mode: Cut the selected field to the clipboard	

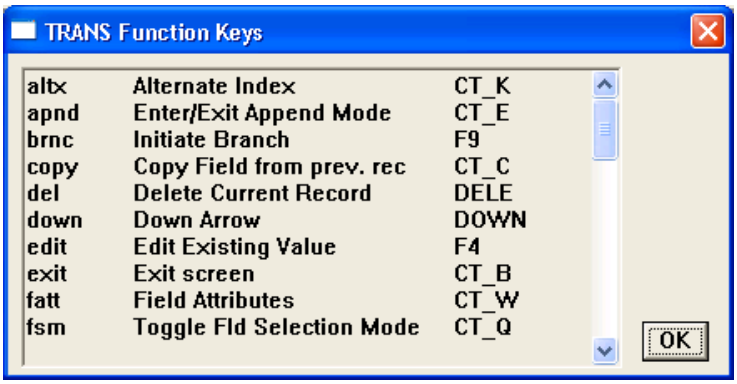
Physical Name[a]	Key Name	TRANS Function Key	Description	F$UNCKEY=physical (if TRANS function key)
CT_Z	Ctrl/Z			
CT_[	Ctrl/[			
CT_\	Ctrl/\			
CT_]	Ctrl/]			
CT_^	Ctrl/^			
CT_ _	Ctrl/_			

a. Physical Name is what the ADMINS reserved field F$UNCKEY contains if the key is not in use as a TRANS function key, while TRANS Function is what F$UNCKEY contains when the key is used as a TRANS function key. This list shows the default use of function keys for TRANS Function. If F$UNCKEY=PHYSICAL is specified in the Trans Environment File the physical key names are used for all keys except control keys (such as ctrl/b) - these keys will be loaded with a two character value (for example "CB" for ctrl/b) if the physical key is mapped to a TRANS function, and a four-character value (the physical key name, for example "CT_J") if the physical key is not mapped to a TRANS function.

b. See Appendix J: "The TED Text Editor" for a description of the TED editor. See Appendix K: "Using Text Fields" for a general discussion of the use of text fields. Section 5.16 "Text Fields" describes how text fields are specified in SCREEN.

6.2.1 TRANS Function Key Help

TRANS displays keystroke function help in a scrollable pop-up window (as shown below) if you press the SHFK key, or click on Help/Function keys in TRANS' Windows Menu Bar. The keystroke help reflects the settings specified in the TRANS Environment file.



6.2.2 KEYPAD Keys

The Keypad keys may be used both as a numeric keypad (NumLock on), or as function keys (NumLock off). This use may be toggled by the NumLock key at any time.

KP_0	Keypad 0
KP_1	Keypad 1
KP_2	Keypad 2
KP_3	Keypad 3
KP_4	Keypad 4
KP_5	Keypad 5
KP_6	Keypad 6
KP_7	Keypad 7
KP_8	Keypad 8
KP_9	Keypad 9
KP_/	Keypad/
KP_*	Keypad *
KP_-	Keypad -
KP_+	Keypad +
KP_E	Keypad Enter
KP_.	Keypad .

6.3 TRANS Modes

TRANS can be in one of four data entry modes: Update, Append, Insert, and Error.

6.3.1 Update Mode

This is the usual mode for TRANS, and it is the initial mode when a screen is activated (except when the main file is empty, in which case TRANS automatically starts in Append Mode if Append Mode is allowed on the screen). During Update Mode, editable and loggable fields can be updated, text fields can be displayed and edited using TED, branches can be made to other screens, records can be deleted, and the user can put TRANS into Insert, Append or Error Mode. In Update Mode, each time the user enters a manual change, the disk block containing the active record is written back to the disk.[2]

During Update Mode a message ("TOF" or "EOF") is displayed on the Status Line to indicate that the first record in the file is being displayed ("top of file") or the last record is being displayed ("end of file"). The "TOF" or "EOF" message overwrites the text in the last 13 characters of the first line of a screen. These message displays are controlled with the MSG keystroke (see Section 6.8 "Control Functions"), or the NOMSG screen instruction file keyword (see Section 5.3.1.2 "NOMSG: Inhibit On-line Messages").

6.3.1.1 Update Mode Under LFEXIT Control

LFEXIT control in Update Mode is used to provide an environment under which data entry can be stringently controlled with a minimum of application programming. When LFEXIT control has been requested[3] for a screen, it is activated when the user enters a value into a non-key field.

Under LFEXIT control, the active record is written to disk only when the user presses NEXT (or otherwise invokes TRANS' NEXT keystroke function) to leave the record. This provides a straightforward method for performing data checkout at end of record processing. In addition, data entry in Update Mode is more efficient because the block containing the active record is written back to the disk only once per record instead of after each field is entered.

Since TRANS under LFEXIT control does not write to the disk on a field by field basis, all field logging is performed at once. All changes to the record are logged when the active record is written to disk (i.e. at the NEXT keystroke). A log record is generated for each loggable ("L" or "LR") field in the active file record whose value has been changed, including fields changed by the RMO.[4]

In Update Mode, ordinarily the user can leave the active record in a file using any of a number of keystrokes (e.g. NREC, PREV, BRNC, EXIT etc.). Once the user enters a non-key field, LFEXIT control prevents the user from leaving the record by any means other than the NEXT function.[5] TRANS normally displays the next record in the file when the user presses NEXT in Update Mode. However, if lowercase "b" is in the string assigned to the logical name OPTION and LFEXIT control is active, then when NEXT is pressed, the record is filed but TRANS remains at the record which was just changed. (The BEGREC RMO calls occur; and the cursor goes to the first

---

2. Under LFEXIT control, or if the NOWRITE keyword is present on the screen header line, the block containing the active record is NOT written back to disk every time manual change is entered. LFEXIT control is discussed in Section 6.3.1.1 "Update Mode Under LFEXIT Control". NOWRITE processing is described in Section 16.1.1 "High Volume Update: NOWRITE".
3. To request LFEXIT control the keyword LFEXIT or LFBACK must be present on the header line of the screen instruction file, or the field names section of the screen instruction file must have at least one REQUIRE statement (see Section 5.5.5 "REQUIRE Statement").
4. Ordinarily (without LFEXIT control), only fields manually updated by the user are logged (see Section 6.5.2 "Field Logging, Method of Operation"). Changes made by the RMO are only logged if they are made after LFEXIT mode is in effect (i.e. after a manual change to the record). Please note that field logging under LFEXIT control ***requires*** that the field MODE/A2 must be present in the field log file (see Section 6.5.3 "Expanded Field Log Facilities").
5. By placing the LFBACK keyword on the screen header line, the screen developer can allow the user to "back out" of the current record (see Section 5.3.1.18 "LFEXIT or LFBACK: Update Mode Control").

editable, lookup, or text field.) Using OPTION "b" the user can stay on the current record even after changes have been filed. Remaining on the record just changed may be useful before a branch using branch fields in the active record.

6.3.2 Append Mode

This mode can be entered (if Append Mode is allowed on the screen (see Section 5.3.1.1 "INSERT, DELETE, or APPEND Records")) by pressing the APND keystroke and is used to append records to the bottom of a file, including initial entry of records into an empty file. (When a screen is activated on an empty file, TRANS automatically goes into Append Mode if Append Mode is allowed on the screen.) In Append Mode the user can only enter values, and use the NEXT keystroke function to file away the record which has been entered on the screen, and get another blank screen in which to fill in another record. (Values on the record being entered can be changed by simply moving the cursor to a field, and entering another value "on top" of the previously entered one. Also, the EDIT keystroke can be used for editing alphanumeric fields in Append Mode).[6] When the APND keystroke is pressed again to leave Append Mode, TRANS is placed in Update Mode at the last record in the file. A handy way to see the last record appended when in Append Mode is to press APND to leave Append Mode, look at the last record, and then press APND again to continue in Append Mode. If APND is pressed while in Append Mode the record on the screen is **not** filed, i.e. the values entered on the "blank form" never are written to disk. Therefore, the operator will normally press NEXT to file the last entered record before pressing APND to leave Append Mode. TRANS will not exit (i.e. via EXIT), or branch from the current screen while in Append Mode.

During Append Mode the message[7] "AP" is displayed on the status line.

6.3.3 Insert Mode

This mode can be entered (if Insert Mode is allowed on the screen, (see Section 5.3.1.1 "INSERT, DELETE, or APPEND Records") via the INS keystroke function to insert another record under an existing key value, or by clicking "Yes" in response to the

prompt presented to the user after a failure to match on an entered key.

6. Text fields in the main file of a screen cannot be displayed or edited (via TED) in Append Mode. As explained in Appendix K: "Using Text Fields", because no actual record exists in Append Mode until the record is filed (via NEXT), text fields in the main screen file cannot be used.
7. The MSG keystroke (see Section 6.8 "Control Functions"), or the NOMSG screen instruction file keyword (see Section 5.3.1.2 "NOMSG: Inhibit On-line Messages"), will suppress the "AP" indicator.

Insert Mode stays active while the inserted record is "filled in", until the NEXT keystroke function is used to file the entered values, i.e. to write them to disk. Then TRANS returns to Update Mode, with the record just inserted active. TRANS will not exit (i.e. via the EXIT key), or branch from the current screen while in Insert Mode.

During Insert Mode the message[8] "IN" is displayed on the status line.

6.3.4 Error Mode

This mode is entered when TRANS detects an entry error due to a format check, or a Data Dictionary validity or codelist check (see Appendix I: "ADD: The ADMINS Data Dictionary"), or when a Check expression evaluates to "true" (see Section 5.5.6 "Check Statement"), or when a REQUIREd field is null at end-of-record processing (see Section 5.5.5 "REQUIRE Statement"), or when the RJ$RJ **local** RMO field is set (see Section 16.1.2 "Reject APPEND, INSERT, UPDATE, DELETE, or Transfer"). In Error Mode the cause of the error (and the error message if its a check statement) are displayed in a pop-up box, and all other keystrokes are ignored until the user clicks "OK" (or hits Enter) to acknowledge the error. Then TRANS returns to the mode it was in before the error occurred.

6.4 Entering or Changing Fields

When data is entered into a non-key field of a record in Update Mode, it is ordinarily written to the disk ("filed") immediately, i.e. the entire block containing the active record is copied from memory to the disk.

In Update Mode under LFEXIT control (see Section 6.3.1.1 "Update Mode Under LFEXIT Control"), the record is not written to disk until the NEXT keystroke function occurs.[9] In this case, the block containing the active record is written back to disk only once for each record, when the updates for that record have been completed.

In Update Mode under NOWRITE control (see Section 16.1.1 "High Volume Update: NOWRITE"), changes to the record are not filed until the RMO running behind the screen causes the record to be written to disk.

In Append Mode, the user fills in the fields of a "blank form". Nothing is written to disk until the NEXT keystroke function occurs. When NEXT occurs, the filled-in active record is appended to the end of the file, and a new "blank form" is displayed.

In Insert Mode, a blank record with the requested key values is written to disk as soon as "I for INSERT" has been entered. None of the values entered into non-key fields are actually written to disk until the NEXT keystroke function occurs.

8. The MSG keystroke (see Section 6.8 "Control Functions"), or the NOMSG screen instruction file keyword (see Section 5.3.1.2 "NOMSG: Inhibit On-line Messages") will suppress the "IN" indicator.
9. Or until the RMO causes an automatic NEXT (B$B = 'LF'), see Section 16.2.2 "Automatic NEXT key: B$B = 'LF'".

If "Y" is assigned to the logical name ADM$READONLY, TRANS opens all files read only. Insertion, deletion, and appending to all files is blocked: nothing can be changed in any file on disk when the screen is in read only mode because TRANS never writes to disk (the cursor will go only to key fields and ALLOW fields (see Section 5.5.14 "ALLOW statement") when ADM$READONLY is in effect).

6.4.1 Keystrokes: Entering or Changing Fields

ENTER (return or enter or tab): Used to enter data into a field. For example, to enter the value "4.50" into the field "DISCOUNT" the operator places the cursor at the DISCOUNT field, types "4", ".", "5", and "0" and then the ENTER keystroke. The ENTER keystroke causes the data to be checked out, entered into the record, and "tabs" the cursor to the next editable, lookup, or text field. (The AUTOCR keyword, see Section 5.3.1.3 "AUTOCR: Automatic Carriage Return", can be used to eliminate the need to press the ENTER keystroke when entering data.) Once you have started typing data into a field, TRANS editing keys may be used.

If the ENTER keystroke is given without any data being typed, then the field is not changed, and depending on the current Field Selection Mode (tabbing or query), the cursor tabs forward to the next field (tabbing) or the user can type a new field selection (query). ENTER is used in Append, Insert and Update Modes.

On Windows, if you use the mouse to click on another object on the TRANS screen, if appropriate, TRANS will act as if you typed ENTER before the click action takes effect, i.e. if a data entry error occurs TRANS will return the cursor to the field that was being entered.

On Windows, the TAB key acts the same as return or enter (invokes the AdmTrans ENTER keystroke function described above).

COPY: (For fields from the main file only). If the first keystroke pressed at the current cursor position is COPY, the value of this field in the previous record in the file is copied to the current record.

PLUS: (For integer fields from the main file only). If the first keystroke pressed at the current cursor position is PLUS, the value of this field in the previous record in the file is incremented by one and then placed in the current record.

LOOK: If the LOOK keystroke function occurs before or during the time a field is being typed in, TRANS will display the LOOKUP file window for that field, if one exists (see Section 5.11 "LOOKUP Window"). When the cursor goes to a field with LOOKUP available, a clickable string "Lkup" appears on the Status line (if you click it the Lookup Window is displayed).

LKDO: If the LKDO keystroke function occurs before or during the time a field is being typed in, TRANS will display the LOOKUP file window for that field, if one exists, in "display-ony" mode, as described in Section 6.11.1 "Display-only LOOKUP".

UP, DOWN, LEFT, RGHT (arrows): In "tabbing" Field Selection Mode the cursor is moved in one of four directions. LEFT and RGHT move the cursor to the previous or next editable, lookup, or text field using the order of the field names from the screen description file as the guide. UP and DOWN move the cursor to the last editable, lookup, or text field on the previous line or the first editable, lookup, or text field on the next line respectively.

EDIT: Ordinarily when you type characters into a field TRANS substitutes the new entry for the old when you press RETURN. TRANS also allows character by character editing of the current value of the field at the cursor. Position the cursor at the field to be edited and press the EDIT keystroke. You can then use typical Windows editing functions to alter the field contents (for example, you can toggle between insert and overstrike mode).

When the cursor goes to a text field, a clickable string "Text" appears on the Status line (if you click it, or press EDIT, the designated word-processor is launched displaying the contents of the text field). Text fields cannot be displayed or edited in TRANS until the word processor designated in the data dictionary (see Appendix I.4.3 "Text Fields") is launched.

6.5 Field Logging

TRANS can be instructed to maintain an automatic log of changes to fields in the master file of the screen. If a field log file was created when the master file was defined (see Section 2.10 "Field Logs"), or if a field log file is explicitly identified in the TRO[10] (see Section 5.3 "Screen Header Line"), then TRANS will automatically maintain a log of changes made to the "loggable" fields in the screen (see Section 5.5.3 "Loggable").

As was shown in Section 2.10 "Field Logs", DEFINE creates two files, TELFON.MAS (20,000 records) and TELFON.FLG (5,000 records), from the following DEF, because the FLGSIZ specification (e.g. 5000) is included. The layout for TELFON.FLG is shown following the DEF:

```
*              TELFON.DEF
*  Telephone directory file definition
MAS  20000  5000
LNAME     A20   KEY1   "last name"
FNAME     A10   ASC2   "first name"
INITIAL   A1            "middle initial"
TELNO     A8            "telephone number"
#STREET   A6            "street number"
STREET    A20           "street name"
CITYST    A30           "city and state"
ZIP       X99999        "zip code"
```

```
*  Layout for TELFON.FLG
CHGDAT DA  ASC1  "change date"
TSEQ   I   ASC2  "transaction sequence #"
DLC    DA        "date of last change"
LNAME  A20       "key field from master file"
FLDNAM A8        "name of changed field"
FLDTYP A2        "data type of changed field (I,D,F,DA,A,X)"
TTYP   A2        "transaction type"
SEQ    I         "sequence for multi-line change"
OLD    A16       "old value"
NEW    A16       "new (changed) value"
```

The field log holds the following information for each change made to a file: the date of the change, the transaction sequence number assigned (automatically) to that change, the date of the last change, the key value(s) for the record being changed, the first eight characters of the name of the field being changed, the data type of the

10. A field log file explicitly referenced in the TRO will override the field log file created when the master file was defined.

changed field, the transaction type, and the old and new (changed) value. If the old or the new value in the field is larger than 16 characters, then a field log record is written for each 16 characters of the field and the field SEQ contains the sequence of the multiple records.[11]

Each file whose definition calls for a field log automatically has fields DLC and TSEQ built into its internal record definition. These two fields , DLC for "date of last change" and TSEQ for "transaction sequence number", are set by TRANS with the DLC and TSEQ of the FLG record that logged the last change to the record. The field log transaction record can in turn be used via its DLC and TSEQ to pick out the previous transaction record that logged the previous change to the original file record, and so on. In this way ADMINS can provide[12] an audit and recovery trail of changes to records made via TRANS. The field log file could be used to produce reports about file activity, or to reprocess record changes (via PROD) against an earlier version of the file, if the latest version of the file were lost.

6.5.1 Field Log Example

To clarify the field log function, the following is an example showing the results of field logging.

Using the following file definition, DEFINE creates DEMO.MAS and DEMO.FLG. The field types were chosen to illustrate the result of a change to each of the six ADMINS data types.

```
*     DEMO.DEF
MAS 1000 100
REC    I   KEY1  "record number"
IFLD   I         "I type field"
DFLD   D2        "Dn type field"
FFLD   F4        "Fn type field"
DAFLD  DA        "DA type field"
AFLD   A30       "An type field"
XFLD   XA9A9A9   "Xpic type field"
```

In addition to the fields in the file definition, DEMO.MAS will also include the following fields because DEFINE was asked to create a field log.

```
TSEQ   I         "transaction sequence number"
DLC    DA        "date of last change"
```

For purposes of illustration, DEMO.MAS has three records and the fields have the following values.

```
REC   IFLD    DFLD      FFLD       DAFLD
---   ----   ------   --------   ---------
  1    123   456.00   789.0000   25-DEC-81
  2    234   567.00   890.0000   01-JAN-82
  3    345   678.00   901.0000   15-JAN-82

REC     AFLD                              XFLD   TSEQ     DLC
```

11. By creating an explicit DEF for a field log file, the size of the OLD and NEW fields in the field log file can be extended up to A80. This method is recommended for handling fields larger than A16. See Section 6.5.3 “Expanded Field Log Facilities” for a discussion of this facility.
12. The log file is a legitimate ADMINS file and can be displayed and analyzed using all the ADMINS tools. Usually there is a transaction log per master file. There may actually be several logs per master file, one for each kind of update or each operator. There is a place in the screen description form for assignment of a particular log for a particular screen.

```
---  ----------------------------  ------  ----  ---------
  1  This is a test of fields logs.  A1A1A1    0
  2  Any alphanumeric data is fine.  B2B2B3    0
  3  The purpose is to see the log.  C3C3C3    0
```

The following shows the field log transaction records that were created by TRANS when changes were made to fields in the records.

```
   CHGDAT   TSEQ     DLC     REC  FLDNAM  FLDTYP
---------  ----  ---------  ---  ------  ------
19-JAN-82     1                1  IFLD    I
19-JAN-82     2                2  DFLD    D
19-JAN-82     3                3  FFLD    F
19-JAN-82     4  19-JAN-82     1  DAFLD   DA
19-JAN-82     5  19-JAN-82     2  AFLD    A
19-JAN-82     6  19-JAN-82     2  AFLD    A
19-JAN-82     7  19-JAN-82     3  XFLD    X

TSEQ  TTYP  SEQ     OLD               NEW
----  ----  ---  ----------------  ----------------
   1  UP      1  123               321
   2  UP      1  567.00            765.00
   3  UP      1  901.0000          109.0000
   4  UP      1  25-DEC-81         15-JUN-81
   5  UP      1  Any alphanumeric  This field gets
   6  UP      2   data is fine.    2 log records.
   7  UP      1  C3C3C3            F6F6F6
```

Finally, the contents of DEMO.MAS after the changes were made. Using the field log transaction record and the before and after representations of the file, you can follow how the fields were changed and the audit trail of the change.

```
REC   IFLD    DFLD      FFLD       DAFLD
---   ----   ------   --------   ---------
  1    321   456.00   789.0000   15-JUN-81
  2    234   765.00   890.0000   01-JAN-82
  3    345   678.00   109.0000   15-JAN-82

REC     AFLD                          XFLD   TSEQ     DLC
---  ----------------------------  ------  ----  ---------
  1  This is a test of fields logs.  A1A1A1     4  19-JAN-82
  2  This field gets 2 log records.  B2B2B3     6  19-JAN-82
  3  The purpose is to see the log.  F6F6F6     7  19-JAN-82
```

6.5.2 Field Logging, Method of Operation

When ever a **manual** change is made to a loggable[13] field in Update Mode, a record describing the change is appended to the field log file. Fields set by an RMO running behind the screen changes are **not** ordinarily logged. However, TRANS will log all changes (manual or set by RMO) to loggable fields when LFEXIT control is active (see Section 6.5.3 “Expanded Field Log Facilities”).

13. Fields are specified as loggable in the screen description file (TRS). See Section 5.5.3 “Loggable”.

6.5.3 Expanded Field Log Facilities

Users may choose to define their own field logs, rather than use the field log created automatically by DEFINE when a field log size is included in the file description line in the DEF. Choosing from the field log fields described above and additional field log fields described below the user can request either a simpler or a more detailed field log than the one which is automatically provided.

The choices available to the user include making the field log a keyed file; not including some of the automatic field log fields, i.e. field type of the logged field; and adding additional fields to the field log to capture such information as last transaction sequence, operator id, time of change, or operator.

TRANS also supports full **record logging** as well as field logging. By placing the field name MODE/A2[14] in the field log file, the user is requesting TRANS to log every **non-null** value that was present in the master file record at the time when a record is appended, inserted or deleted. In addition placing the field MODE/A2 in the field log file enables field logging under LFEXIT control (see Section 6.3.1.1 "Update Mode Under LFEXIT Control") and with NOWRITE (see Section 16.1.1 "High Volume Update: NOWRITE"). (Changes made to records in Update Mode without NOWRITE or LFEXIT control are logged as in simple field logging.)

Below we list the automatic field log "DEF".

```
FLG flgsiz
*
CHGDAT  DA  ASC1  "date of change"
TSEQ    I   ASC2  "transaction sequence"
DLC     DA        "date of last change"

master file keys

FLDNAM  A8        "name of logged field"
FLDTYP  A2        "type of logged field"
TTYP    A2        "transaction type"
SEQ     I         "multi-record sequence number"
OLD     A16       "string representation of old value"
NEW     A16       "string representation of new value"
```

If the user creates an explicit DEF for the field log file then any of these fields may be omitted, their order may be changed, and any fields may be made keys. The user may **not** change the field types except that the user may change the **length** of the fields OLD and NEW. The length of the fields OLD and NEW may be up to A80 (both must be the same). This facility can be used to eliminate the multi-record field log transactions (i.e. SEQ greater than 1).

The following fields may also be included in a field log DEF.

```
LSEQ  I   "last transaction sequence for the master file"
TTNO  A2  "terminal number, e.g. B2"
OPER      "operator id"
TIME  A8  "time of logging"
MODE  A2  "mode, requests record logging"
```

Each time a field log entry is made the TSEQ and DLC fields on the active record are set to the field log record number and the current date (i.e. TSEQ and CHGDAT in the field log). Before changing the TSEQ and DLC fields in the master file record these values are placed in LSEQ and DLC in the field log record, creating a chain of

14. AdmScreen checks that the field log file contains the field MODE if the TRS specifies LFEXIT or NOWRITE, and the TRS has loggable fields. If the log field cannot be found, or it does not contain the field MODE non-fatal error message "scr505" will be printed warning that no logging will occur.

pointers to all field log records for a given master file record. (If the field log file is defined explicitly, the TSEQ and DLC fields **must** be included explicitly in the master file DEF.)

The OPER field in the field log is set from an actual or virtual field called OPER in the master file record. The OPER field can be of any type, but should have the same type in both the master file and the field log file.

6.6 Record Moving and Searching

When TRANS is called, if there are records in the file it is to utilize, TRANS goes into Update Mode on the record at the top of file.[15] In Update Mode, the user may move around in the file from record to record by several different means. A common method is for the user to enter values into the key field or fields. Once one field of the key is entered all other editable fields in the key must also be entered. Entering a key requests a record search for a specific record to be displayed on the screen.

When a record search on key value fails to find a matching record there are three possible responses from TRANS.

1. The record with the nearest smaller key value is found and displayed. This occurs if neither the INSERT nor the MATCH keywords were present in the screen instruction header line.

   If "k" (lowercase) is included in the string assigned to the logical name "OPTION" (see Appendix A), TRANS, by default, displays the record with the nearest larger key value.

   The screen header line keywords PREV and NEXT (see Section 5.3.1.13 "PREV, NEXT: Record to Display if Key not Found") are used to modify the default action for the current screen. If PREV is present TRANS will always display the previous record, if NEXT is present TRANS will always display the next record, irrespective of the option "k" setting.
2. The MATCH keyword (see Section 5.3.1.8 "MATCH: Require Exact Match") instructs TRANS to go to the top of file. The MATCH keyword in the screen header line requires the match of an entered key value to a record in the file. If no match occurs TRANS goes into Error Mode (See Section 6.3.4 "Error Mode"). When the error is cleared TRANS goes to the record at the top of the file.
3. TRANS offers to insert a new record with the sought after key value. If the INSERT keyword was present in the screen description, TRANS displays the pop-up "OK to Insert" box. If the operator does not respond "Yes" then the record of nearest (higher) key value(s), (or the last record in the file) is displayed and TRANS stays in Update Mode. If, on the other hand, the operator does respond "Yes", a record containing the sought after key value and blanks or zeroes in non-key fields is inserted in the file, and is displayed on the screen. TRANS enters Insert Mode on this new record. The operator may then enter values for the non-key fields. The values are **not** filed one by one. Rather, they will all be filed (i.e. written to the disk) when the operator signifies the completion of the entries for the new record. This is done when the NEXT keystroke function occurs, which is the only way to exit Insert Mode. NEXT

15. TRANS can be instructed to enter the file on a specific record. See Section 6.15 "Entering TRANS On A Specific Record".

performs an update of the complete new (inserted) record before returning to Update Mode on the just inserted record. (The blank record with key values set **is** inserted into the disk file when the user types "I", before the user starts entering data.)

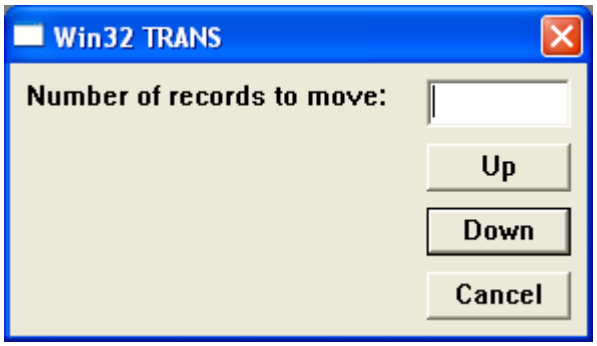
- **HOME:** This keystroke moves the cursor to the highest editable key field. When values have been entered into all the key fields (there may be just one), the implied instruction for TRANS is to search for and then display the record with the just entered key value(s). As was explained above, whenever a value is entered into the first editable key field values **must be entered** into all of the subsequent editable key fields.
- **NEXT:** This keystroke causes display of the next record in single record screens and of the next page in multi-record screens. In Append Mode, Insert Mode, and Update Mode under LFEXIT control, NEXT causes the active record to be written to disk.
- **NBRK:** When BREAK is in effect on a multi-record screen (see Section 5.9.2 "BREAK In a Multi-Record Screen") this keystroke starts the next display page at the next control break. Otherwise, this keystroke acts the same as NEXT in most cases, i.e. it causes display of the next record in single record screens and of the next page in multi-record screens. Unlike NEXT, however, NBRK **does not** cause the active record to be written to disk in Append Mode, Insert Mode, or Update Mode with LFEXIT control.
- **PREV:** This keystroke causes display of the previous record in single record screens, and of the previous page in multi-record screens. In Update Mode the PREV keystroke can be used to "back out" of updates to a record when LFEXIT control is active. No updates to the current record are written to disk.[16] This "backout" function is enabled if the LFBACK keyword is present on the screen header line (see Section 5.3.1.18 "LFEXIT or LFBACK: Update Mode Control"). When PREV is entered under LFEXIT control, TRANS prompts as follows:

  ```
  "PRESS PREV TO CONFIRM BACKOUT"
  ```

  The user enters PREV again to confirm that the updates are not to be filed.
- **PBRK:** When BREAK is in effect on a multi-record screen (see Section 5.9.2 "BREAK In a Multi-Record Screen") this keystroke starts the display page at the previous control break. Otherwise, this keystroke acts the same as PREV in most cases, i.e. it causes display of the next record in single record screens and of the next page in multi-record screens.

16. This facility is useful for the situation when LFEXIT control is being used to ensure that only complete, valid records are being filed; but the user does not have enough information to complete a valid record. After backing out, the user can continue entering valid records, coming back to the problem record when complete information is available.

- **NRECS:** Displays a pop-up box that prompts for the number of records to move:



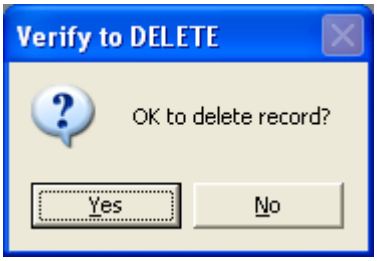
Just click "Up" or "Down" to move to the records at the beginning or the end of the file. Enter a number, then click a direction button, to move that number of records toward the beginning or end of the file.

- **ESCAPE**: Hitting the ESCAPE key to cancel the typing in a field and restore the original value.

6.7 Record Operations

- **APND:** This keystroke is used to enter entirely new records which will be appended to the end of the active file. Once in Append Mode, i.e. after pressing the APND keystroke, the operator is presented with a blank form. The UP, DOWN, LEFT, and RGHT (arrow) keystrokes can be used to move the cursor among the fields on the form. Once the cursor is positioned at a field, data can be entered there. Data can also be overwritten (i.e. corrected) on a field by field basis on the "append" screen. The particular record being entered is **filed** when the operator presses the NEXT keystroke. After filing that record the screen is cleared of data and another record can be appended. During Append Mode most other function keystrokes are disabled. To leave Append Mode the operator presses the APND keystroke once again (if Append Mode is exited before the NEXT keystroke, the record is not filed). The last record in the file is displayed and TRANS is once again in Update Mode. The APND keystroke is ignored if the APPEND keyword is not present on the header line of the screen instruction file.
- **DEL:** In order to delete a record, the record must first be brought onto the screen either by moving to it or by searching it out. When the DEL keystroke function occurs must be pressed, TRANS will check, and ignore the DEL keystroke, if the current record is the only record in the file, or the only record in a locked range (see Section 5.5.1.1 "Restrict TRANS to Key Range").TRANS

displays a "Verify to DELETE" pop-up.

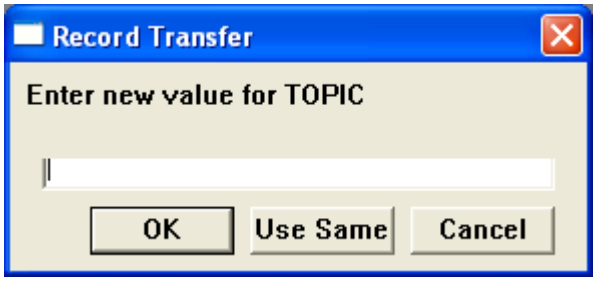


Click "Yes" (or press Enter[17]) to delete the record. The DEL keystroke is ignored if the DELETE keyword is not present on the header line of the screen instruction file. After deletion of a record TRANS displays the next record in the file.

- **INS:** This is the INSERT keystroke. It is used to insert records that have key values that already exist in the file. (ADMINS supports multiple records with the same key values.) To insert another record with the same key value after the record being displayed, enter INS. If the INSERT keyword is active for this screen, TRANS will enter Insert Mode and the entire insert sequence will ensue, as described under variation (3) of inserting after a failure to find a key match (see Section 6.6 "Record Moving and Searching"). During Insert Mode most other function keystrokes are disabled.
- **TR:** This keystroke is used to transfer records. A transfer can be thought of as a delete followed by an insert, or as a **refiling** of information in the current record under a new identification (key value). This function is only enabled if DELETE is permitted on the screen. After typing TRF, the operator enters a new key value indicating the position in the file to which the current active record is to be transferred.

The entire transfer sequence occurs as follows:

1. The operator positions TRANS at the record to be transferred.
2. The operator presses the TRF keystroke. TRANS will check, and ignore the TRF keystroke, if the current record is the only record in the file, or the only record in a locked range (see Section 5.5.1.1 "Restrict TRANS to Key Range").
3. If DELETE is permitted on the active screen, then TRANS prompts with a pop-up:



for new key values.[18].

During the record transfer dialogue, the "Use Same" button can be used to copy the value for a particular key from the current record to the new record.

17.Place *"delete.default=N" i*n the TRANS environment file to make the "No" button the default button in the "Verify to DELETE" pop-up

4. The operator enters new key values under which the active record is to be filed. If a record with these key values does not already exist then the key values in the active record are changed to these new key values and the active record is refiled under these new key values. If a record with these new key values does exist, then the operator is asked whether the active record is to be filed first or last in the sequence of existing records that already have the key value. The operator responds and the active record is refiled.

6.8 Control Functions

- **EXIT:** When this keystroke function occurs TRANS displays the Exit confirmation pop-up:

The user can click “Cancel” to return to the screen session, “Yes” to display a File Open dialog box for a new TRO, or “No” to exit the TRANS session. The EXIT keystroke function can be inhibited by the NOEX keyword on the screen header line as described in Section 5.3.1.16 “NOEX: Inhibit Screen Exit”.

PRT: .This keystroke copies the current screen image to the clipboard and then presents a printer dialog box to select a printer to print the image.

- **MSG:** This keystroke toggles the display/hiding of standard informational messages on the status line. These messages include field format prompting, and the "top of file" (TOF), and "end of file" (EOF) messages. The NOMSG instruction in the TRS screen header line starts TRANS with these messages hidden.
- **FSM:** This keystroke changes the Field Selection Mode between "query" and "tabbing". In "query" field selection mode a popup box allows you to select from a list of field names available in the current screen. When a selection is made,the cursor will move to that field to accept an entry for that field.

  In "tabbing" field selection mode the operator presses ENTER or one of the four directional arrows to tab the cursor from field to field.

18. If TRANS is in a locked range of keys (see Section 5.5.1.1 “Restrict TRANS to Key Range”), you cannot transfer the record out of the locked range. TRANS only asks for the keys "below" the locked range, i.e. if KEY1 and KEY2 of the file have been locked into a range TRANS will only prompt for KEY3, KEY4 etc.

6.9 Branching and Subscreens

To perform a branch select from the sub-menu of available branches.

1. This sub-menu is displayed by any of the following actions:
   a. Select “Branch...” in the File Menu
   b. Press the BRNC keystroke.
   c. Click on the BRANCH icon : 

   After execution of a branch TRANS displays the target screen with the target record and the user may then use all the facilities of TRANS under the control of the target screen.
2. The XRET keystroke function is used to return to the screens that are the "sources" of the last several branches. TRANS keeps track of the sequence of branches in a "stack". As you branch from screen to screen in the TRANS session, information about each screen you leave is added to the top of the stack. (The number of past branches that can be tracked is limited, and depends on the key size of the main files involved in the branch. About 10 branches can usually fit on the stack. When the stack overflows, older branches are removed from the bottom of the stack to make room for the latest branches on the top. If XRET is used to return to a screen, TRANS displays the record that was active when the branch was made from the earlier screen. (Using BRNC, TRANS would display the record with the key values formed by the branch fields.)
3. The XFWD keystroke function is used to retrace “forward” through screens visited via XRET.

For example, lets assume you start in screen A and branch successively to screens B, C, D, and E. Assume further that you move around to different records in each of the screens before you make the next branch. From whetever record you were on in screen E you could use XRET (3 times) to return to screen B. The same record that was active when you branched from screen B to screen C would be the active record when you return to screen B.

You could then return to the record you left in screen E by pressing XFWD 3 times.

The XRET  and XFWD  toolbar icons can also be used to invoke XRET and XFWD. The XRET icon is enabled once a BRANCH occurs in the TRANS screen session, and the XFWD icon is enabled once XRET occurs.

6.9.1 Subscreens

A screen can consist of a main screen and up to 14 subscreens. (Subscreens are described in detail in Section 5.15 “Subscreens”.) When a particular subscreen is active, TRANS behaves as if the screen consists only of the fields in the main screen and that subscreen. The Subscreens sub-menu displays when the SUBS keystroke function occurs, or when you click on “Subscreens...” in the File Menu.The Subscreens sub-menu looks and works like the branch menu (see Section 6.9 “Branching and Subscreens”). The main screen always has "subscreen code" 0, while subscreens are numbered from 1 to 9 and A to E.

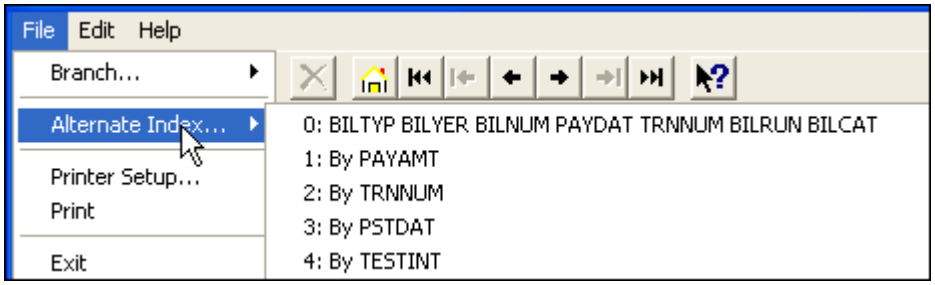
Using the menu, the user can go to any subscreen in the current screen, or to the main screen (by itself).

All TRANS features and keystrokes are available within subscreens. Specifically, since the key fields should be in the main screen, the user can enter a key value to go to another record without changing subscreens.

6.10 Changing the Active Index

With ADMINS for Windows a TRANS user can switch to any index[19] active for the file, unless the necessary fields are not present in the screen, or the developer has restricted or disabled this capability[20].

If enabled, the ALTX keystroke, or clicking on the "Alternate Index..." entry in the File Menu, pops the Alternate IndexSub-Menu
:



The numbers and names of all the indexes available for the main file of the current screen are displayed. Disabled indexes and indexes that cannot be used[21] are shown "grayed". When an index is selected from the list it becomes the "active index" which will be utilized to perform all record movement and search operations.

For example, you might have a file with employee biographical data that has a primary index that provides access to employee records by employee ID number, and alternate indexes that access the records by name, or by department and postion number.

Changing the active index does not change the active record. If you use the employee ID index to find employee ID "8654" with name "ADAMS" and then switch to the name index you will still be looking at the record for employee 8654/ADAMS. But when you then press the NEXT keystroke, you would move to the next record in the name index (perhaps employee ID 1263, name AGGANIS) rather than employee ID 8655 (the next record in the Employee ID index).

19.See Section 2.7 "Multiple Indices"
20.See Section 6.17.9 "Suppressing Items in the File Menu"
21.Indexes cannot be used if all the keys for that index are not displayed on the the screen, or when BREAK or DL fields are in use, only indexes that have the exact same keys (down to the BREAK or DL field) as the current index can be used.

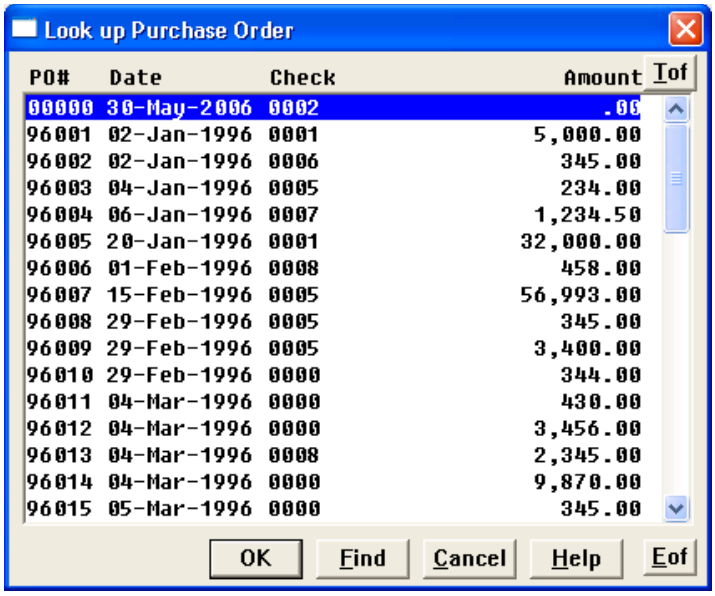
6.11 Lookup Windows

When the cursor goes to a field that has LOOKUP available, the LOOKUP icon

 is activated and the clickable string "Lkup" appears on the Status line. Click "Lkup", or the LOOKUP icon, or press the LOOK keystroke, to activate the LOOKUP window.[22]

If multiple LOOKUP windows have been specified (see Section 5.11.2 "LOOKUP Menu") for the current field, a LOOKUP Menu is displayed. Choose a LOOKUP window by clicking on it, or typing its number, or use the UP and DOWN arrow keys to move to an item, then press ENTER to select it. To back out of the LOOKUP Menu without any action taken press ESCAPE.

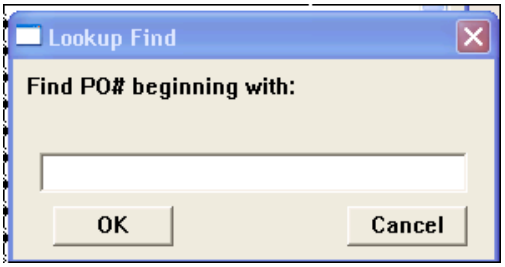
The LOOKUP window provides a scrollable, clickable listing. Select an item by double clicking on it, or select the highlighted item by clicking OK (or pressing Enter).



When an item is selected, the LOOKUP window is cleared from the screen and the RETURN field value from the LOOKUP file (if any) is "entered" into the editable field on the screen. Any TRANSFERs specified in the LOOKUP paragraph are also performed.

22.if the TRANS_ENV keyword ***lookup.rightclick*** is in effect, right-clicking on a field will activate lookup. Right-clicking a field where the TRANS cursor is (the field that has the focus) has the same effect as pressing the ***look*** key. Right-clicking a field not currently in focus has the same effect as left-clicking on the field (puts it in focus) followed by pressing ***look***.

If you press the Find button LOOKUP will prompt for a key value(s).

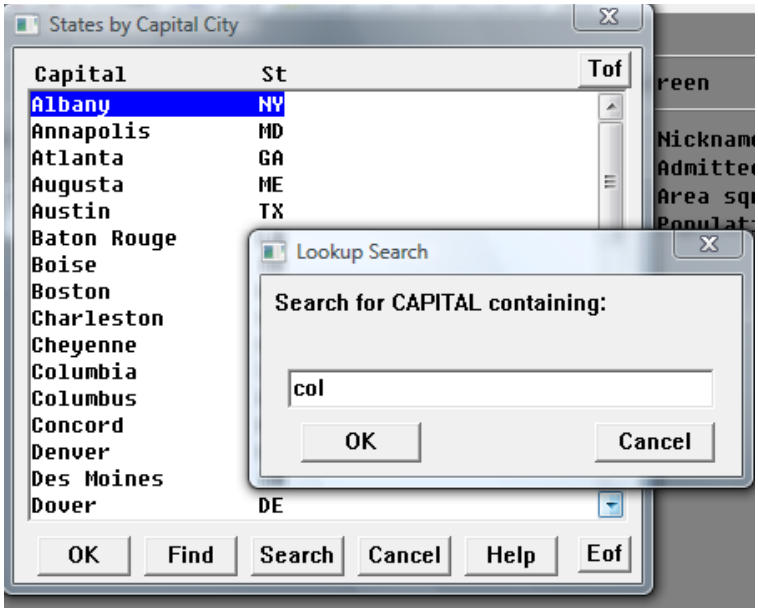


When you have finished specifying the search keys, TRANS will position the LOOKUP window at the best available match (the next previous record) in the LOOKUP file.

Lookup has an optional ***Search*** button[23]. ***Search*** displays records where the alphanumeric key contains the target string in any position

When enabled, ***Search*** will be grayed (disabled) if the first key value prompted for is not alphanumeric.

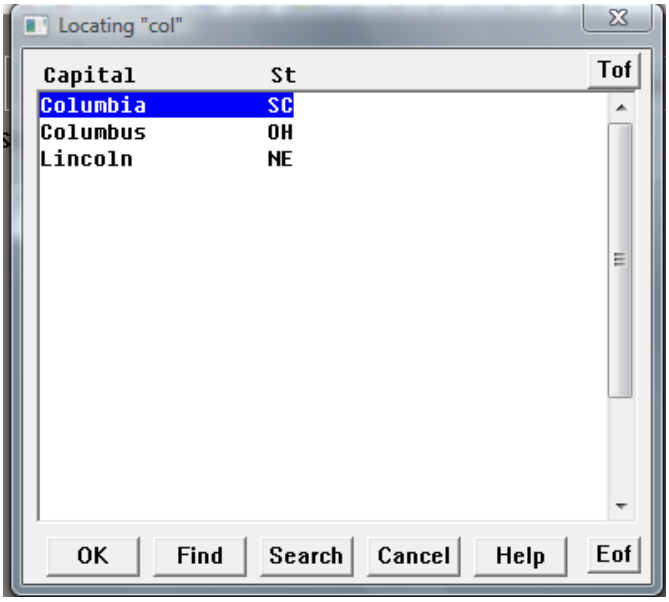
If you click **Search**.



If the first key being prompted for is an A field, you'll get a pop-up prompt:

23. The "Search" button in previous builds of AdmTrans has been renamed ***Find,*** as it is used to find records by key values. ***Find*** appears by default in Lookup windows. See Section 5.11 "LOOKUP Window" for details on enabling ***Search***.

If you type a value then click ***OK*** lookup will move to the top of file (or top of the active range) and start selecting records where the key field contains the entered value in any position. This search is case blind.



Note that ***Find*** would display"Columbia" and "Columbus" but ***Search*** also displays "Lincoln" because it looks for the target string "col" in any position in the key field value.

If the lookup has a SELECT clause this will be applied to any records that pass the search criteria, and only records that pass both criteria will be selected.

When the lookup window is actively searching the lookup window title will be changed to

Locating **<target string>**

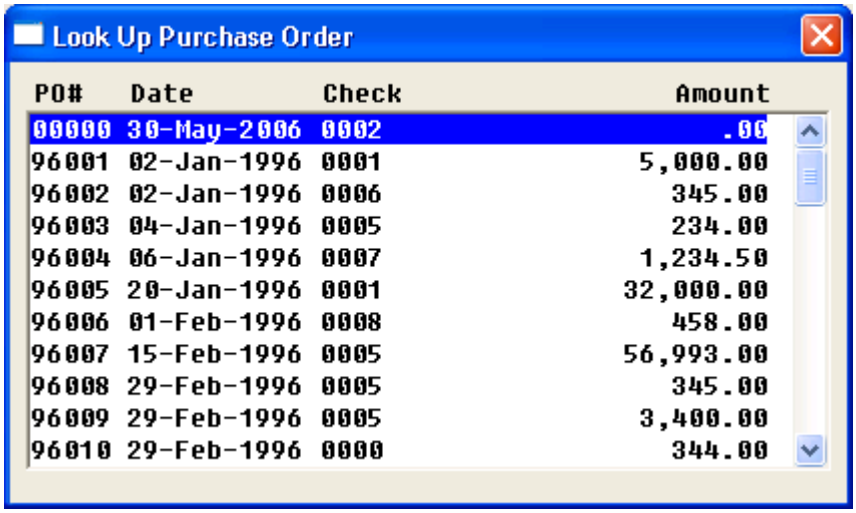
The lookup window title will be reset to its original value when Search is no longer active.

If you click ***Search*** but then click ***Cancel*** the display will be restored to display all records (that pass any SELECT logic). Also, if you click ***Find*** followed by any keystroke and then click ***OK*** any search logic will be cancelled.

6.11.1 Display-only LOOKUP

Lookup windows can be opened in "display-only" mode by using the "**lkdo**" TRANS function key (by default assigned to Ctrl/U). When this key is pressed in a field that has a lookup associated with it, the lookup will pop up and the asked-for records will be loaded and displayed as in a regular Lookup, but the focus will immediately return to TRANS proper, leaving the now out-of-focus lookup window still visible. Return or transfer statements are ignored if the lookup is activated using this keystroke, and the Lookup window's function buttons (OK, Cancel etc.) are not displayed.

.

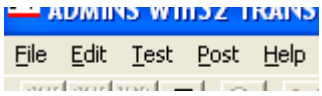


The user may click on the lookup window, move it around, and use the scroll bar to view records not currently visible, but only records loaded when the lookup was activated will be available (the lookup files are closed, only the records loaded into the lookup listbox can be viewed). A typical use of this kind of lookup would therefore be to load all records in a certain key range.

The lookup window will automatically be closed down when a new lookup is activated, or TRANS is exited[24]. The user may also close the lookup window at any time by clicking on the [X] in the upper right corner of the window, or via the "**lkcl**" keystroke (by default assigned to ctrl/D).

6.12 Menu Bars and Submenus

The TRANS window includes the standard Windows Menu Bar:



You can click directly on an item in the Menu Bar, or activate the bar via the Menu keystroke. By default. the Menu Bar contains three items, the "File", "Edit", and "Help" sub-menus. If Menu Bar items are specified (see Section 5.12 "Menu Bar") for the current screen they appear between the Edit and Help items (in the example above items "Test" and "Post" are specified).

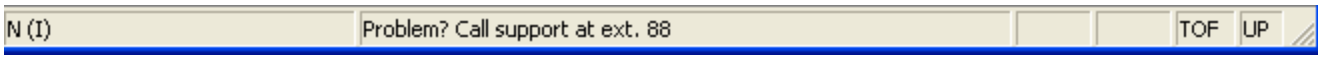
24.The TRANS Environment file entry:

lookup.DisplayOnly=CloseAtEofrec

specifies that display-only lookup automatically closes at any EOFREC call. (this means TRANS will destroy the window at any self-branch and as you move from rec to rec in a multi-record screen.)

6.13 Status line

The TRANS window includes the standard Windows status line



In addition to its standard status line display items (field format information, position in the file, TRANS mode), TRANS will display the contents of the logical name ADM_TRANS_MESSAGE on the status line. In the example above, the "Problem?..." message results from the ADM_TRANS_MESSAGE logical name. ADM_TRANS_MESSAGE is translated and displayed immediately after the first pre-link RMO call in every screen, so the RMO can set it with CRLOG if desired (see Appendix H.9.1 "CRLOG - Create or Delete a Logical Name").

Alternatively, the special local RMO fields M$MSG/An (message text) and M$LOC/I (optional line number for message) can be used to create a message in the RMO and to place it anywhere on the screen. If the TRANS_ENV file contains the entry:

```
m$msg.position=S
```

the contents of M$MSG will be displayed on the status line. Whenever the RMO changes the value of M$MSG, the status line is re-displayed (see Section 16.11 "Status Line Control: M$MSG and M$LOC").

Finally, the reserved field X$MSG[25] (or its global equivalent, the X$MSG TRANS$ENV keyword[26]) can be used to set a value in the left panel of the status bar.

6.14 HELP in TRANS

TRANS provides an on-line help facility. Text files containing any desired information can be displayed in windows from 3 to 24 lines long, at any point in a TRANS session. The help text may be scrolled up and down to display more text than can fit in the window. The format of a help file for TRANS is a text file. The help file contains named sections. Each section starts with a line of the following format.

```
NAME
```

NAME is how this section is referenced from TRANS. NAME must be one word (no embedded blanks) and begins at the left margin.

A given section may be given several different NAMEs by having several different (unindented) NAME lines precede a HELP section.

The rest of the section contains free text of any length, **indented at least one space** on each line. Help files may contain indirect (@@) references.

For example, the following is a section from a help text file.

```
FUND 16 24
The fund number is a 6 digit code. The first digit
indicates the fund type. Valid fund types are:
```

25.See Section 5.5.8.12 "X$MSG: Set Message in Status Bar"
26.See Section 6.17.10 "X$MSG: Set global value for status bar message"

```
1=Government  2=Proprietary  3=Fiduciary  4=Special
The second digit represents the fund sub-type.
...
```

Help is available from any screen in any mode (update, append, insert) via the HELP keystroke. When help is requested, TRANS does the following.

1. First TRANS must find the help file. If the logical name ADM$HELPFILE is assigned, TRANS attempts to open ADM$HELPFILE. If this file is found it will be used. Otherwise, if the logical name ADM$HELPDIR is assigned, TRANS attempts to open ADM$HELPDIR:screen_name.HLP. Screen_name is the name of the currently active screen (note that this is the **screen name,** the first element on the screen header line, not the TRO name).
2. Once TRANS has found the help file, TRANS needs to locate a help section name. If the field H$ELPNAME/An exists and is not blank, its value is used as the section name. Otherwise, TRANS will use the name of the field at the current cursor location.
3. TRANS reads the help file, searching for the requested section. TRANS performs partial matching on the section name (for example, if "ACC" is in H$ELPNAME, TRANS stops searching when it finds "ACCOUNT_NUMBER", which begins with "ACC").

A section name of ANY will always be found, regardless of what section name TRANS is searching for. ANY can be used to provide an error message or other default user instructions. If ANY is used, it should be the last section in a help file. (Otherwise, help sections below it will not be found.)

If the help section name is not found in ADM$HELPFILE, TRANS will search for it in ADM$HELPDIR:screen_name.HLP, if that file exists. If the section cannot be found in either file, TRANS beeps with the message "No HELP for '<help_name>'".

If the requested help section name is found, TRANS displays as much of the section as possible in the help scrolling region.

6.15 Entering TRANS On A Specific Record

TRANS can be used on a command line to call up a specific screen on a specific record indicated by its key value(s). The syntax is:

```
$ trans tro-name/screen-name key_values
```

The "/SCREEN-NAME" is optional, as are the "KEY_VALUES". For example:

```
$ trans org 100

$ trans acctg/expled 100 0531 101

$ trans payrol/pers "KAETZEL"
```

If the specific record indicated is not found, TRANS will go to the next highest key value or the record at the end of the file. If the MATCH function is in effect (See Section 5.3.1.8 “MATCH: Require Exact Match”), when the indicated record is not found TRANS will go into error mode, and then to the record at the top of the file when the error is cleared.

The key-values element will be treated as a logical name if it starts with "A_". For example, if "306469037" were assigned to the logical name A_KEY then TRANS would go directly to that employee record.

```
> admlcr a_key 306469037
> trans payroll a_key
```

Key values can also be supplied via "substitutable parameters" in an ADMINS command procedure , or via environment variables used in either a procedure or on the command line. For example, say we have a file keyed by employee last name, and a personnel screen that displays information on an employee. If we set up a ”.bat” file that calls a screen with a parameter, i.e.

```
> trans payrol/pers %1
```

and we call it “pers.bat”:

then personnel information on any particular employee could be accessed by typing "PERS NAME" as follows:

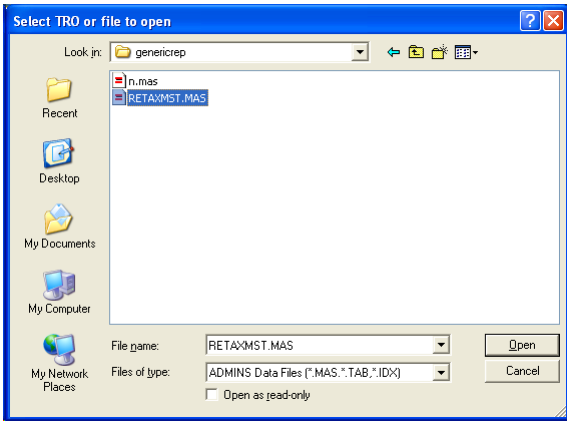
```
> pers "KAETZEL"
```

The personnel information screen, showing the record for "KAETZEL", would be displayed.

6.16 General Editor Mode (“GENED”)

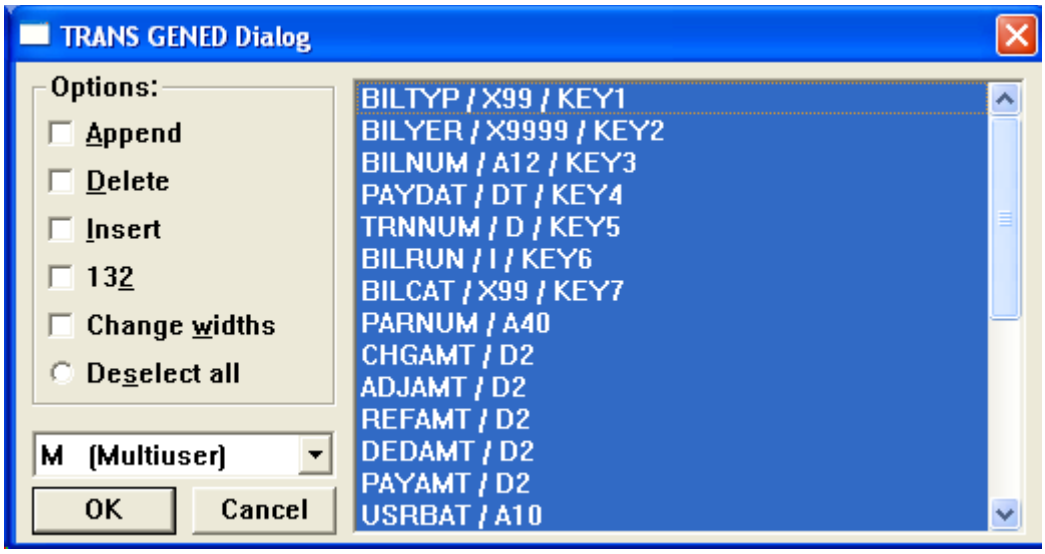
The General Editor Mode is an option in TRANS that can be used without prepared screens as a utility to input, patch or browse through ADMINS files. The General Editor Mode is not intended for general use, but rather as a tool for the applications developer to provide quick and easy access to on-line files.

If AdmTrans.exe is associated with the extensions of the ADMINS data files (usually “.MAS”, “.IDX”, “.TAB” etc.) TRANS will open in GENED mode if you just enter the file name at the command prompt, or if you click on the file entry in a folder. If you just type “admtrans” at the command prompt, or click on admtrans.exe (or a shortcut to it) in a folder TRANS will display a file open dialog box:



to enter GENED mode use the “Files of type” dropdown to select “ADMINS Data Files...” and select a file to open (or just type in the file’s name at the “File name” prompt).

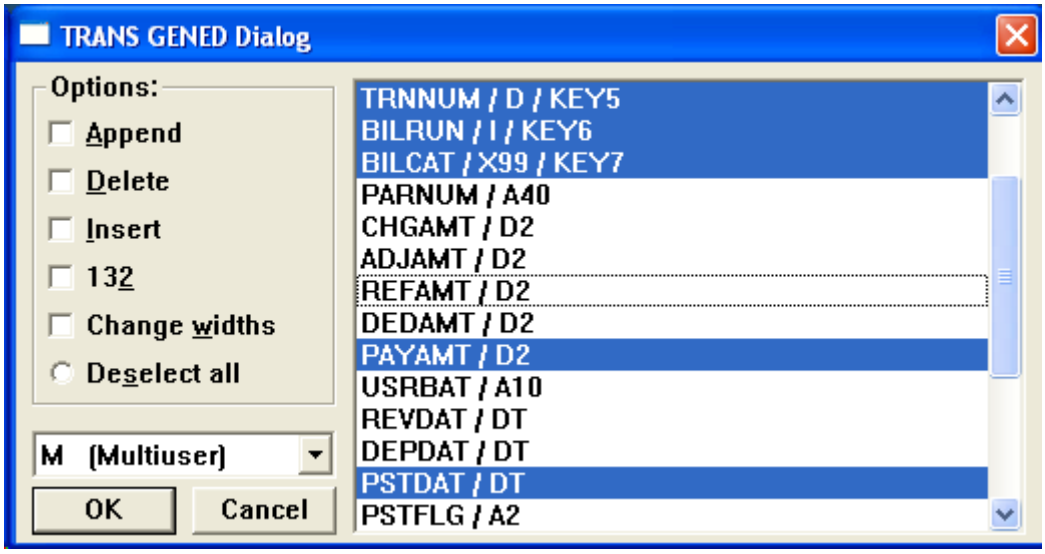
TRANS opens the data file and presents the TRANS GENED Dialog:. :



where you can specify whether to enable Appending, Deleting, and Inserting records in this session; whether the display should be 80 (the default) or 132 characters wide; which fields you want displayed; whether you want to alter the display width of any fields; and the file access mode you want.

6.16.1 Selecting Fields For Display

All fields are selected for display when you start. Key fields are always selected (you cannot deselect them). To deselect all fields click the option in the left pane. Clicking on fields in the right pane toggles between selecting (highlighted) and omitting fields for display.

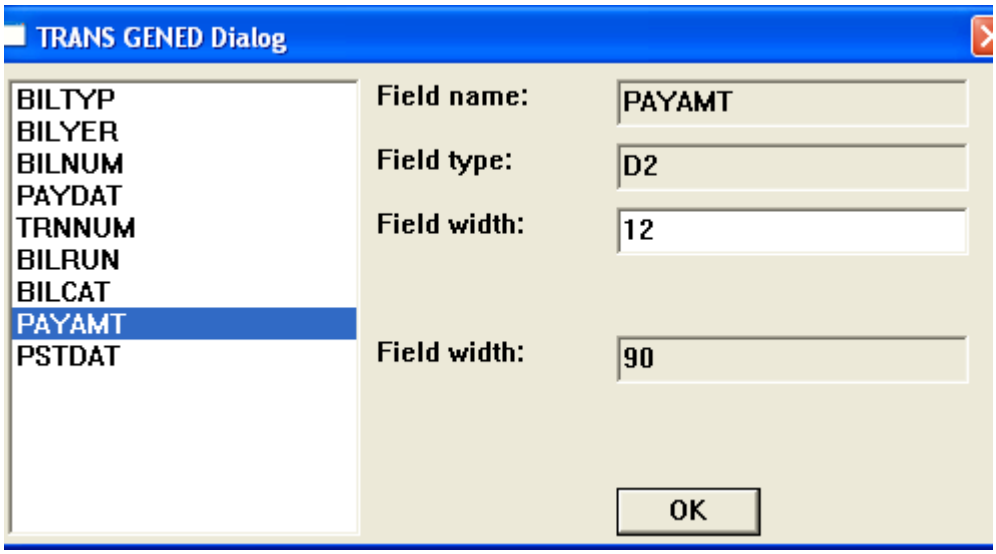


6.16.2 User Specified Field Widths

TRANS uses default widths for the fields as is shown in the following table:

Field Type	Default Width
I - Integer	7
Ln - Longword Decimal	15
Dn - Decimal	21
Fn - Four word decimal	26
DA - Date	9
DT - Date	9
TM - Time	11
An - Alphanumeric	characters in field (n)
Xpic - Picture	characters in picture (pic)

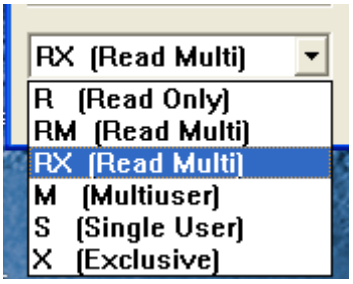
The user can override these widths for the selected fields by clicking “Change widths”, then clicking the OK button (or press Enter):



The above window displays. Select a field from the list on the left and alter its “Field width”. The total display width of all selected fields is displayed below. If the total display width of the selected fields is less than the screen width you have specified (80 or 132) GENED will display a “multi-record” format (i.e. one record per line in the display), otherwise one record is displayed at a time.

6.16.3 File Access Mode

Select the File Access mode using the dropdown box on in the left pane of the Trans GENED dialog window:



The default file access mode in GENED is **Multiuser**.

6.16.4 Alternate Indexes in GENED

If a file has alternate indexes, and you have opened the file via its main index, you can switch to an alternate index using ***Files/Alternate Index...*** on the Menu or via the ALTX keystroke.

If a file is opened in GENED mode directly via an alternate index the key fields of that index are moved to the front of the field list in the GENED dialogue box, and to the top of the edit list when the GENED screen displays, followed by any other selected fields. In this case switching indexes is disabled.

6.17 The TRANS Environment File

Use the TRANS **Environment File** to alter or extend the built-in TRANS environment.

TRANS uses the logical name TRANS_ENV[27] to find the environment file. You may use a relative pathname when specifying TRANS_ENV

You can also specify a particular Trans Environment File to use for a particular screen by placing a TRANS_ENV statement (see in the screen instruction file (TRS). The file named in a TRANS_ENV statement overrides the setting of the TRANS_ENV logical name.

The TRANS Environment File is a text editable file where each line modifes a particular property of TRANS' appearance or behavior. Lines that begin with "!" or "*" are ignored and thus can be used for comments.

The general format is:

```
keyword=value
```

where keyword identifies the specific property we want to affect, and value defines the new setting for the property identified by keyword.

Here is an example of a TRANS Environment File for Win32 TRANS:

```
field.edit.background=blue
window.font=ANSI_FIXED_FONT
dmap:091=198
dmap:092=216
dmap:093=197
```

This TRANS Environment File tells TRANS to display editable fields with a blue background, uses the ANSI_FIXED_FONT as its primary font, and displays the characters '[', '\' and ']' as 'Æ', 'Ø' and 'Å'.

The keywords used in the TRANS Environment File are often constructed as a hierarchy of keywords, separated by dots (.). In our example above,

```
field.edit.background= first
```

specifies the field class of objects, then narrows this to editable fields, and finally specifies the background color property for editable fields. The value part then specifies the new setting for the property or object identified.

6.17.1 Reassign Key for TRANS Keystroke Function

To reassign a TRANS keystroke function to a different standard function key, use the following syntax:

```
%trans_key=KEYSTROKE
```

27. When translating the logical name TRANS_ENV TRANS first checks the process logical name table, then checks the desktop table, then checks the system table.

where:

trans_key	is the TRANS keystroke function name (see Section 6.2 "Standard Functional Keystrokes")
KEYSTROKE	is the standard function key name.

When a TRANS keystroke function is reassigned, the default standard function key for that TRANS function is "freed up" for use at the application level. For example, the following assignment reassigns the TRANS EXIT function to the CT_X(Ctrl_X) key.

```
%exit=CT_X
```

It also allows the default keystroke for EXIT (ctrl/b) to be used for other application purposes (like any other standard function key that is not mapped to a TRANS function).

To disable a TRANS standard keystroke function leave out the standard function key name, e.g.:

```
%exit=
```

disables the TRANS EXIT function (and also frees up ctrl/b for another purpose).

6.17.2 Define Macro Function

To extend the TRANS environment, i.e. to add macro function keys, use the **define** keyword:

```
define func_name=KEYSTROKE sim_key_1 sim_key_2 ... [\~]
```

where:

KEYSTROKE	is the standard function key name that will invoke the macro function. func_name is a name you supply for the macro function, e.g. "get_out" for a function to exit TRANS environment completely.
sim_key1 sim_key2 etc.	are the keystroke or keystrokes that are to be simulated, i.e. the "macro function". A single character (separated by blanks) represents that character. A string of more than one character represents a standard function key. A string that begins with "%" is a function name (either a TRANS keystroke function or a macro name defined previously in the environment file). A macro that contains only a string of the form "M$M_xx" (where "xx" is number between 1 and 99, e.g. MM_23, MM_4) defines a special RMO call. If the defined key is pressed TRANS will call the RMO with M$M set to "xx" (e.g. "23" or "4"). and S$S set to the field where the cursor is located.
\~	denotes the definition is continued on the next line.

Some examples:

To define CT_X to type the word "Canceled" followed by a carriage return:

```
define cancel=CT_X C a n c e l e d CR
```

To define F11 to cause the RMO to be called with M$M set to 23:

```
define call23=F11 M$M_23
```

6.17.3 Rename Standard Function Key

To specify a new name for a standard function key, use the **rename** keyword:

```
rename KEYSTROKE NEW_NAME
```

where:

KEYSTROKE	is a standard function key name.
NEW_NAME	is the new name for the key.

This is useful for development of keyboard independent applications, e.g. you can write F$UNCKEY logic in the record maintenance procedure that references a key named NEW_NAME; and run the application with any keyboard by using "rename" to give an appropriate standard function key on that keyboard the name NEW_NAME.

6.17.4 DMAP and MAP

The **dmap:** and **map:** keywords are used to transpose character values when they are moved between internal storage in ADMINS files and the outside world.

dmap: is used to change a character value when it is moved from internal storage to the outside world, e.g. displayed in a window. This is especially useful when you are using a character set for display where certain characters have a different ASCII code than the character set used for internal storage. For example, in Scandinavia certain 7-bit ASCII characters are used to represent local characters that in most character sets are found in the extended 8-bit character set.

map: is used to change character values when they come from the outside world (e.g. typed at the keyboard) to be stored internally.

The syntax for both keywords is:

```
dmap:sss=ttt
map:sss=ttt
```

where '**sss**' is the three digit decimal representation of the source character, and '**ttt**' is the three digit representation of the target value.

A typical display remap sequence for Norway and Denmark would be:

```
dmap:091=198    !  [ => Æ
dmap:092=216    !  \ => Ø
dmap:093=197    !  ] => Å
dmap:123=230    !  { => æ
dmap:124=248    !  | => ø
dmap:125=229    !  } => å
```

and for Sweden:

```
dmap:091=196    !  [ => Ä
dmap:092=214    !  \ => Ö
dmap:093=197    !  ] => Å
dmap:123=228    !  { => ä
dmap:124=246    !  | => ö
dmap:125=229    !  } => å
```

6.17.4.1 RTFMAP

The first time a piece of internal text is accessed under Win32 ADMINS, its format is converted from the TED WPT format used under OpenVMS to RTF. You may at the same time ask to convert 7-bit codes to 8-bit codes. This is done by placing the

```
rtfmap:sss=ttt
```

keyword in the TRANS_ENV file, using the same syntax as for map and dmap. If you want to use the same mapping as for regular data, you may specify:

```
rtfmap:map
```

instead of having to repeat all the mapping statements.

6.17.5 F$UNCKEY=PHYSICAL, Load F$UNCKEY with Physical Key Names

By default, when F$UNCKEY[28] is present in TRANS' virtual record, it is loaded with the TRANS function name (in lowercase), if any, of the function key that is pressed. If the function key has no TRANS function mapped to it, then the standard function key name of the key is loaded.

If the statement

```
f$unckey=physical
```

is present in the TRANS environment file, TRANS will **always** load the standard function key name into F$UNCKEY, whether or not the keystroke has a TRANS function mapped to it. This feature allows older ADMINS/V32 applications that use F$UNCKEY to run without change with the newer multi-platform versions of ADMINS.

28. See Section 16.15 "F$UNCKEY - Function Key Detection in RMO"

6.17.6 SETKEY=PHYSICAL, Simulate VT-type Function Keys

If the statement

```
setkey=physical
```

is present in the TRANS environment file, TRANS checks the integer array given as an argument to SETKEY[29] to see if it contains any escape sequences that match those sent by VT keyboard special function keys. If so, TRANS will simulate those special function keys when TRANS is called.

This feature allows older ADMINS/V32 applications that use SETKEY to run without change with the newer multi-platform versions of ADMINS.

If the special RMO integer field ADM$NOPHYSICAL is present in TRANS virtual record, the current screen ignores SETKEY=PHYSICAL in the TRANS environment file, allowing applications written using the newer syntax to run correctly even when SETKEY=PHYSICAL is present.

6.17.7 Global Timeout

Specifies a global timeout statement (see Section 5.5.20 "TIMEOUT statement").

global.timeout=seconds actions ...

seconds	the number of elapsed seconds for timeout to occur (seconds must be >= 600 (10 minutes) and <= 7200 (2 hours)
actions	are a list of TRANS keystroke functions (see Section 6.17.2 "Define Macro Function").

Examples:

```
global.timeout=1200 %exit
```

Timeout after 20 minutes (1200 seconds), and exit.

```
global.timeout=1200 %brnc T
```

Time out after 20 minutes (1200 seconds), and branch to branch-code T.

This capability can easily be combined with ***global.branch*** (described in the next entry) to implement a branch to a "safe" branch target (i.e. no records locked, no file conflicts) at timeout.

After this kind of timeout the user could return to the location where the timeout occurred by simply using the %xret keystroke.

29. See Appendix H.13.14 "SETKEY - Simulate Keystrokes in TRANS"

6.17.8 Global Branch

Create a globally recognized branch target (see Section 5.7 “Branches”)..

global.branch=branch_code tro-name/screen name branch-description

Example:

```
global.branch=T SYOBJ:SYHold/HOLD "General Holding Screen"
```

which makes the SYOBJ:SYHold/HOLD screen available in every screen as branch target T.

6.17.9 Suppressing Items in the File Menu

The Branch, Subscreen and Alternate Index menu items, located under the File menu, can be suppressed by entering the following lines in the TRANS_ENV file:

```
menu.file.branch=off
menu.file.subscreen=off
menu.file.altix=off
```

If one of these File menu items is suppressed the keystroke that activates that menu item is disabled (as there is no menu to display). However you can utilize these keystroke function, using SETKEY or a PUSHBUTTON action macro[30] to activate a particular item (i.e. a specifuc branch, a specific subscreen, or a specific alternate index). For example:

```
BP ix1 'Switch to Index 1'
  action=%altx 1
BP br1 'Return to screen origin'
  action=%brnc X
```

These two PUSHBUTTONS would allow, respectively, a switch to alternate index 1 and a branch to branch-code “X”, even when display of the “Alternate indexes...” and “Branch” sub-menus has been suppressed.

This implementation allows the developer to provide controlled access to branches, subscreens, and indexes when unlimited manual selection by the user is not desireable.

6.17.10 X$MSG: Set global value for status bar message

To avoid having to individually specify each screen where the X$MSG[31] status bar message is desired, the TRANS_ENV file can be used to declare a flexible global X$MSG message.

In the TRANS_ENV, declare a line:

```
X$MSG=Text to display
```

30.See Appendix H.13.14 “SETKEY - Simulate Keystrokes in TRANS” and Section 5.5.21.1 “PushButton Object”

31.See Section 5.5.8.12 “X$MSG: Set Message in Status Bar”

where "Text to display" is a mixture of literal text and field names enclosed in % signs, e.g.

```
X$MSG=Added %ENTDAT% by %entusr% Changed %cngdat% by %CNGUSR%
```

If the screen's virtual record (TRO) contains the field X$MSG its value takes precedence over this TRANS_ENV setting.

6.17.11 NOL$PROMPT - Don't prompt for L$ parameters (text initialization)

ADMINS internal text fields have the ability to specify an initialization file that governs the creation of the initial text the first time the file is opened. One way to get data into the the text is using L$ parameters (for example <L$AMOUNT>, which will be replaced by the value of the logical name L$AMOUNT). Usually the L$ logicals have been set (perhaps by the RMO) when the user presses the EDIT key on an empty text field.

By default, if the logical name does not exist, the user will be prompted for a value to be substitued for the parameter. To change this behavior use the keyword

```
NOL$PROMPT
```

on a line by itself in the TRANS.ENV file. L$ parameters for which no logical name exists will not result in prompting the user for a value (the parameter will be ignored).

6.17.12 "Localizing" Messages and Prompts

Messages and prompts in TRANS are "soft"; they can be customized (or "localized") by the application developer, for example, to appear in a different language, or to reflect the organization's terminology.

In TRANS, this can be implemented[32] via TRANS_ENV file commands in the form:

```
msg###=message
```

where ### is a three digit decimal number identifying the message[33]. To replace TRANS message number four, use:

```
msg004=Message Four
```

To conserve trailing blanks in a message, put the message in quotes, e.g.

```
msg007="Trailing blanks "
```

6.17.13 TRANS_ENV Lexicon

The following symbols and conventions are used to describe TRANS_ENV options and syntax:

- ***PKY*** - Physical keyname
- ***MB#*** - Mouse Button Number (i.e. MB1, MB2)
- %func - TRANS keystroke function ("%exit", "%prev")
- xvalue|yvalue - alternation: either xvalue or yvalue
- [value] - optional, may be omitted
- ***string*** - string of ascii characters ("abc") and/or octal codes for ascii characters ("\111\112\113")
- *brush* - the background color of the trans window - can be one of the following: white, lightgray, gray, darkgray, black or hollow.
- *font* - may be given as one of the system fonts, or as a point size and a font name. The system fonts are **ANSI_FIXED_FONT** or **SYSTEM_FIXED_FONT**. By default TRANS uses the **SYSTEM_FIXED_FONT** to display literal text and field values. The **ANSI_FIXED_FONT** is a little smaller, and may be preferable for users with small screens or using a screen resolution less than 800x600. **Font** may also be given as

  ```
  point-size,"font name"
  ```

  For example:

  ```
  window.font.132col=8,"Courier New"
  ```

  uses the default font (SYSTEM_FIXED_FONT) for 80 column screens, and an

32. Localization can also be implemented using the ADM$LOCALE file, see Section 1.9 "Localizing ADMINS".
33. The list of messages and prompts is contained in the file adm$dist:admins.msg. In this file TRANS message number 4 is tra004; to change tra004 in the TRANS_ENV file use the command "msg004=".

8 point Courier New font for 132 column screens (observe that the "" are mandatory if the font name contains a space). Be aware that only a syntax error gives an error message. Asking for a font name that does not exist, or a point size that is not available for the requested font results in Windows providing the closest match it can obtain.

- *color* - may be specified as a color name, or as an RGB value[34]. E.g. the two entries:

```
field.display.background=skyblue
field.display.background=135  206  235
```

  specify the same color. Not all display units and video cards are able to distinguish between all color settings, so you might have to experiment to find the right color combination for your display.
- *macro* - Series of keystrokes, can be character, ***PKY*** or %func

6.17.13.1 TRANS main program

map:xxx=yyy	remap ASCII decimal code xxx to yyy
dmap:xxx=yyy	remap (display only) xxx to yyy
define fname = ***PKY*** macro	define keystroke macro
rename PKY newname	rename physical key
f$unckey=physical	F$UNCKEY loaded with physical names rather than TRANS function names
setkey = physical	Numeric **SETKEY** values are interpreted as physical escape sequence values. Only VTxxx escape sequence values are recognized.s
global.timeout	Specifies a global timeout statement (see Section 6.17.7 "Global Timeout").
global.branch	Create a globally recognized branch target (see Section 6.17.8 "Global Branch").
cutoffyear=YY	When a date is entered with a 2 digit year, all years < YY is interpreted as 20YY, and all years >= YY as 19YY.
insert.default=No	Makes the **No** button the default button in the **OK to insert?** dialog.
delete.default=N	Makes the No button the default button in the Verify to DELETE pop-up.
print=clipboardonly	Forces Ctrl/P to go only to the clipboard, and not be printed.
print=landscape	Forces Ctrl/P to print in landscape mode.
print=default	Forces Ctrl/P to use the default printer.

34. RGB values are given as three (3) numbers in the range 0-255 identifying the intensity of red, green, and blue. Thus '0 0 0' (no color) specifies black, and '255 255 255' (all colors, full intensity) specifies white. (The three color codes may be separated by whitespace, or a comma followed by zero, or more whitespace).

print=landscape,default	Forces Ctrl/P to print in landscape mode using the default printer
print.dimension=x,y	Ctrl/P output will be dimensioned to x by y pixels.
print.command=***Command***	Ctrl/P will use ***Command*** to print the content of the clipboard instead of the default 'TedRe /PC'
weekstart=monday	To make the week start with Monday in the date field automatic lookup calendat.
lookup.rightclick	Enables right-clicking on a field to activate lookup. Right-clicking a field where the TRANS cursor is (the field that has the focus) has the same effect as pressing the ***look*** key. Right-clicking a field not currently in focus has the same effect as left-clicking on the field (puts it in focus) followed by pressing ***look***. If the field ADM$MPOS (see Section 16.13 "ADM$MPOS: Detecting Right Mouse Button") is present in the RMO, the RMO will get its usual call (M$M set to 'FX' and F$UNCKEY set to 'msf3'). ***Lookup.rightclick*** **is ignored** if the RMO takes some action at this call, e.g. branching, activating a subscreen, moving the cursor, or simulating a keystroke by calling SETKEY. The developer can ensure that ***lookup.rightclick*** will be ignored at this RMO call by setting F$UNCKEY to "DONE". Setting F$UNCKEY = 'SKIP' tells TRANS to ignore the right-click completely. Setting F$UNCKEY = 'DONE' tells TRANS to perform normal 'msf3' processing, but to ignore the ***lookup.rightclick*** behavior. In most cases TRANS will be able to detect that the RMO took some action so ***lookup.rightclick*** will be ignored anyway, but setting F$UNCKEY = 'DONE' makes this unconditional.
lookup.menu=AlwaysDefault	Highlight and always select the first lookup in a lookup menu if no other lookup is selected. A side effect of this is that you cannot escape from the lookup menu without selecting a lookup.
lookup.MenuOnCancel	TRANS returns to the lookup menu if the user clicks on the Cancel button in one of the lookups chosen from the menu.
lookup.DisplayOnly=CloseAtEofrec	Display-only lookup (see Section 6.11.1 "Display-only LOOKUP") automatically closes at any EOFREC call. (This means TRANS will close the window at any self-branch, or as you move between records in a multi-record screen.
menu.file.branch=off	Disable branch sub-menu in File menu (also disables branch (brnc) keystroke and branch toolbar button).
menu.file.subscreen=off	Disable subscreen sub-menu in File menu (also disables subscreen (subs) keystroke).
menu.file.altix=off	Disable alternate index sub-menu in File menu (also disables Alternate Index (altx) keystroke).
OrigKB=ExtKB	AdmTrans will make no distinction between the function keys on the numeric keyboard and the corresponding function keys on the extended keyboard. If this keyword is present, pressing leftarrow on the extended keyboard and pressing 4 on the numeric keyboard (with NumLock off) will have the same effect.

recordlock.ask=once	If the user chooses "Ignore" to the `Record Locked by Another User` message AdmTrans will go into Read Only mode, and no write back to any files will take place thereafter, until you move on to another virtual record with no ignored record locks (e.g. by hitting Enter or branching to another record/ screen).
NoFileHelp="Path of WinHelp file"	TRANS will call WinHelp with the specified filename if the field help lookup fails. (Defines a default help file).
nol$prompt	Internal text initfile functionality does not prompt for a value if an L$ value is not found.
%msf1=***[MB#]***	Mouse fld-fld movement: Set/disable mouse button
%***tfky***=[***PKY***]	Set/disable the trans function ***tfky*** mapped to physical key ***PKY***. For example, to have the trans function key "exit" mapped to the physical key "ctrl \" (control + backslash): **%exit=CT_** The above entry would also "free up" the CT_B keystroke for another use in the application. To disable a trans function keystroke, don't assign it to a physical key, e.g: **%trf=** The above entry would disable the "transfer" keystroke and "free up" the CT_T keystroke for another use in the application. See Section 6.2 "Standard Functional Keystrokes" for a list of trans function keystrokes and physical keys, showing the default mappings.

6.17.13.2 WINDOW keywords

The window keywords assign properties that affects the background window in which all TRANS objects are displayed. TRANS currently recognizes the following keywords:

window.background=brush|color	background color for TRANS window
window.foreground=color	color for screen literals
window.font=font	main (default) font for screen (literals and data)
window.font.132col=font	main (default) font for 132-wide screen
window.font.80col=font	main (default) font for 80-wide screen
window.font.button=font	font for push buttons: The button font may also be given as ANSI_VAR_FONT (which will also be the default button font if the window font is given as ANSI_FIXED_FONT).
window.font.checkbutton=font	font for check boxes

window.font.radiobutton=font	font for radio buttons
window.font.buttonhovertext=font	font for button hover text
window.font.hovertext=font	font for field hover text
window.font.##=font	declare up to 30 fonts for use in screen
window.style=maximized	start screen maximized
window.caret=vbar	use vertical bar as TRANS cursor
window.caption=addusername	add user name to window caption (title)
window.startposX=pixel	specify (in pixels, from left) horizontal position of screen on desktop
window.startposY=pixel	specify (in pixels, from top) vertical position of screen on desktop

window.font.##= allows you to specify up to 30 fonts, numbered 20 through 49. These fonts are intended for use with LABEL statements in the TRS, and for buttons.

For example, if you have

```
window.font.30=6,"Verdana"
```

in the TRANS_ENV file you can reference that font as number "30" in a LABEL statement (see Section 5.5.15 "LABEL Statement") or the BUTTON subroutine (see Section H.13.3 "Button - Creating and Modifying Buttons in TRANS").

6.17.13.3 BOX properties

Thess settings affect boxes drawn by TRANS as a result of box statements in the .TRS (see Section 5.5.10 "BOX statement"). The keywords available are:

box.background=brush|color	brush or color for box background (brush values are the same as for window.background
box.foreground=color	color for box foreground
box.edge=type	specify the type of edge used to frame the box. The valid edge types are: bump, etched, raised, sunken, and none.

box.background entries only affect the background of fully drawn boxes (rectangles).

6.17.13.4 FIELD keywords

In Win32 TRANS, **fields** are a window class by themselves, and may be assigned a variety of properties. Some of these properties can be assigned in the .TRS source file for the screen, or manipulated via the RMO running with TRANS, when the screen is instanciated (see Chapter 5: "AdmScreen: Compiling Screen Forms"). Some of the field properties may be given default values through the TRANS Environment File. Currently the following **keywords** are recognized:

field.border=*number*	border size for all fields (all sides)

field.xborder=*number*	size of "x-axis" or horizontal (top/bottom) border for all fields
field.yborder=*number*	size of "y-axis" or vertical (left/right) border for all fields
field.edge=*type*	used to specify the type of edge used to frame all fields. Valid types: bump, etched, raised, sunken (the default), and none.
field.key.background=*color*	background color for key fields
field.key.foreground=*color*	foreground color for key fields
field.edit.background=*color*	background color for editable (E,ER,L,LR) fields
field.edit.foreground=*color*	foreground color for editable (E,ER,L,LR) fields
field.display.background=*color*	background color for display (D,DR) fields
field.display.foreground=*color*	foreground color for display (D,DR) fields
field.display.text=nocursor	Prevent the cursor from going to text fields (TI fields) when the text field is in display-only mode (e.g. D TEXTFIELD in the TRS).
field.focus.background=*color*	background color for the field currently in focus (where the insertion cursor is)
field.focus.foreground=*color*	foreground color for the field currently in focus (where the insertion cursor is)
field.label.background=*color*	background color for label fields (e.g. designated in COLOR LABEL statement)
field.label.foreground=*color*	foreground color for label fields (designated in COLOR LABEL statement)
field.msg.background=*color*	background color for message (M) fields
field.msg.foreground=*color*	foreground color for message (M) fields
field.color_NN.background=color	background color for field color-scheme NN
field.color_NN.foreground=color	foreground color for field color-scheme NN

The ***number*** in **field.border=** specifies the number of pixels used by TRANS to frame, or place a border, around each field on the screen. For example, three (3) pixels put a distinctive border around the fields. Be aware that this frame takes real estate on the screen, so a narrow border results in a smaller overall window. The **field.xborder** and **field.yborder** keywords may be used to assign different border values for the x and y axes. By default, 1 pixel is used for the x-axes, and 2 pixels for the y-axes.

The **field.label** keyword specifies values for fields used as target fields in a **DDATTR** call to obtain field labels, thus making it possible to display such fields with much the same attributes as literal text.

In addition user defined field color-schemes may be defined:

```
field.color_NN.background=color
field.color_NN.foreground=color
```

Where 'NN' is a number between 20 and 99 (i.e. 80 user defined color-schemes may be defined).

To designate which fields use which color-schemes use the COLOR statement in the TRS (see Section 5.5.16 "COLOR Statement") .

These color schemes may also be set using the special fields H$NAME and H$CODE (see Section 16.5 "Highlighting Fields")

6.17.13.5 Multi-Record Screen keywords

By default, when a multi-record screen does not display a full page of records, the "empty" fields for the partial page do not appear in the TRANS window. You can change this behavior so that the empty fields do appear by placing "mulrec.noclear" in the TRANS Environemnt File.

In a multi-record screen it is possible to have the field in the current active multi-record highlighted differently than the fields from the inactive records. This can be done by simply changing the background (or foreground) color of the fields, or each individual field types (key, edit, display) can be given individual highlighting that differs from the standard highlighting.

The keywords available are:

mulrec.noclear	Fields remain visible for empty positions when multi-record screen does not display full page of records..
mulrec.background=color	background color for active record
mulrec.foreground=color	foreground color for active record
mulrec.key.background=color	background color for key fields of active record
mulrec.key.foreground=color	foreground color for key fields of active record
mulrec.edit.background=color	background color for editable fields in active record
mulrec.edit.foreground=color	foreground color for editable fields in active record
mulrec.display.background=color	background color for display fields in active record
mulrec.display.background=color	foreground color for display fields in active record

The mulrec.background= and mulrec.foreground= keywords may be used to set the background and foreground color for all field types. For example, if your normal field background is white, specifying mulrec.background=lightblue causes all fields in the currently active record to be light blue.

Please note: Any "mulrec." highlighting entries must appear **after** any "field." highlighting entries in the TRANS_ENV file.

6.17.13.6 Displaying TRANS Messages (two types)

message.background=color	background color for %MESSAGE messages
message.foreground=color	foreground color for %MESSAGE messages
m$msg.background=color	background color for m$msg messages

m$msg.foreground=color	foreground color for m$msg messages
m$msg.position=S	Display the M$MSG field messages in the window status line.
m$msg.size=n	When displayed in the window status line, the default size of the M$MSG display area is 40 fixed width characters. Use this option to specify a different size (allowable range 20-80 inclusive)

See Section 5.13 "The MESSAGE Facility" for a description of $MESSAGE messages.

See Section 16.11 "Status Line Control: M$MSG and M$LOC" for a description of M$MSG messages.

6.17.13.7 "Video" Highlighting in TRANS for Windows

The traditional highlighting schemes from traditional character cell terminals: **bold**, underline, **reverse** and blink are implemented as different background and foreground color schemes. The keywords, and default values, are:

highlight.bold.background=color	background color for bold (default is white)
highlight.bold.foreground=color	foreground color for bold (default is darkred)
highlight.underline.background=color	background color for underline (default is gold)
highlight.underline.foreground=color	foreground color for underline (default is darkred)
highlight.bold+underline.background=color	background color for bold+underline (default is white)
highlight.bold+underline.foreground=color	foreground color for bold+underline (default is darkgreen)
highlight.blink.background=color	background color for blink (default is black)
highlight.blink.foreground=color	foreground color for blink (default is gold)
highlight.underline+blink.background=color	background color for underline+blink (default is black)
highlight.underline+blink.foreground=color	foreground color for underline+blink (default is lightgreen)
highlight.bold+blink.background=color	background color for bold+blink (default is gold)
highlight.bold+blink.foreground=color	foreground color for bold+blink (default is darkgreen)
highlight.bold+underline+blink.background=color	background color for bo+ul+bl (default is lightgreen)
highlight.bold+underline+blink.foreground=color	foreground color for bo+ul+bl (default is darkred)

There is no specific setting for **reverse** (i.e. "reverse video"), as the current background and foreground colors will be reversed.

The video keywords may be abbreviated to two characters, i.e. **bo, ul** or **un,** and **bl** (same as for video attributes in SCREEN and TRANS video attributes), for example, the following two statements are equivalent:

```
highlight.bold+underline.foreground=yellow
highlight.bo+ul.foreground=yellow
```

6.17.13.8 Viewtext

vu.mode=edit[a]	Allows the VIEWTEXT file to be edited.
vu.mode=edit,ask	If the VIEWTEXT does not exist the user will be asked if he/she wants to create the file.
vu.mode=edit,new	If the VIEWTEXT file does not exist, an edit window will automatically be opened allowing the user to enter text to create the file.
vu.buttonOK=text	Text to display on the VIEWTEXT OK button.
vu.buttonNext=text	Text to display on the VIEWTEXT Next button.
vu.buttonPrev=text	Text to display on the VIEWTEXT Prev button.
vu.buttonSearch=text	Text to display on the VIEWTEXT Search button.
vu.searchText=text	Text to display in the Search Dialog Box.

a.All the vu.mode=edit keywords require that the pathname of the editor to be used is assigned to the logical name ADM$TEXTEDIT. E.g.:

```
> AdmLcr ADM_TEXTEDIT notepad
```

6.17.13.9 AdmTed Editor Keywords

There are a number of keywords you may put in the TRANS_ENV file to adjust the behavior of **AdmTed.exe** when used to edit/view internal text from TRANS:

admted.fontsize=n	where **n** is any valid point size for the default font used by AdmTed when editing internal text.
admted.nol$prompt	suppresses the prompting for missing **L$** parameters in initialization template files when automatically initializing a new piece of internal text.
admted.nominimized	disables the minimize option in the AdmTed editing window.
admted.save_changes =n	where **n** can be any number between 20 and 4000, to have the text automatically saved after **n** keystrokes (by default, internal text is saved after 1000 keystrokes).

6.17.13.10 Toolbar

AdmTrans (by default) displays a toolbar on top of the screen, allowing many TRANS functions to be performed by a mouse click

The toolbar can be customized by using the "toolbar.default=" keyword in the TRANS_ENV file. .

toolbar.default=[btn,btn...]	specifies (or disables)the toolbar
toolbar.strings=ON	specifies that buttons should display text (as well the button icon)

The standard (default) toolbar would be specified[35] as:

```
toolbar.default=22,25,26,14,-1,12,-1,5,-1,15,20,16,18,19,17,21,-1,11
```

where the positive numbers are from the list below, and -1 represents a separator (used to group toolbar buttons).

```
 1: "Copy"
 5: "Delete"
 6: "Insert"
11: "Help"
12: "Lookup"
14: "Print"
15: "Home"
16: "PBrk"
17: "NBrk"
18: "Prev"
19: "Next"
20: "Tof"
21: "Eof"
22: "Branch"
25: "Back"
26: "Forward"
```

The keyword

```
toolbar.default=
```

with no arguments disables the toolbar.

6.17.13.11 User Defined Toolbar Buttons

The developer may add customized Toolbar Buttons in the TRANS_ENV file. The general syntax is:

```
Toolbar_NN=ICON#; TEXT; TOOLTIP TEXT; ACTION CODES
```

where:

NN	is a number between 70 and 99 (numbers below 70 are reserved for TRANS internal use). This number must be used in a toolbar.default=statement to include this button in the current toolbar.
ICON#	is a number that identifies the icon to be displayed in the toolbar button.
TEXT	is the short text (normally one word) to be displayed in the button when toolbar.strings=ON is active.
TOOLTIP TEXT	is the explanatory "hover" text that appears when the mouse cursor rests on the toolbar button.
ACTION CODES	is a list of standard TRANS action codes to be executed when the button is pressed. A maximum of 15 action codes may be specified.

Built in Toolbar Icons:

TRANS has two classes of built in Toolbar Icons: The standard Windows Toolbar Icons, and ADMINS TRANS Standard Toolbar Icons. They are:

35.The standard toolbar appears by default without the need for any entry in the TRANS Environment File. The example shows how to specify a toolbar that would be equivalent to what TRANS displays by default.

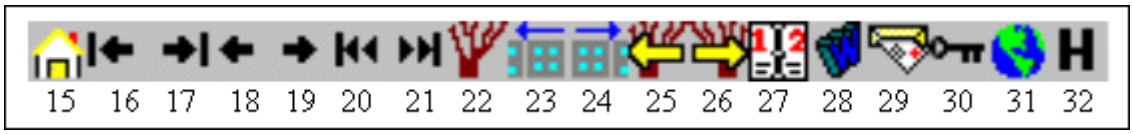
Windows Toolbar Icons:

Bitmap#	Description	Used by TRANS Toolbar	Icon
0	Cut		
1	Copy		
2	Paste		
3	Undo		
4	Redo		
5	Delete	Delete Record	
6	New File		
7	Open File		
8	Save File		
9	Print Preview		
10	Properties		
11	Help	Help	
12	Find	Lookup	
13	Replace		
14	Print	Print	

ADMINS Toolbar Icons:

Bitmap#	Description	Used by TRANS Toolbar
15	Home	Home
16	Previous Break	Previous Break
17	Next Break	Next Break
18	Previous	Previous

Bitmap#	Description	Used by TRANS Toolbar
19	Next	Next
20	Top of File	Top of File
21	End of File	End of File
22	Branch	Branch (New Screen)
23	Previous Window	
24	Next Window	
25	Back	Previous Screen (xret)
26	Forward	Forward Screen (xfwd)
27	Calendar	Lookup
28	Text	
29	Mail	
30	Key	
31	Map	
32	History	
33	Calculator	



6.17.13.12 APPMENU keywords

The following keywords are used in AppMenu:

appmenu.userprofile=**pathname**	specifies the path and name of the AppMenu User Profile File.
appmenu.taskfile=**pathname**	specifies the path and name of the AppMenu Task File.
appmenu.menuspec=**pathname**	specifies the path and name of the AppMenu Menu Specification File.
appmenu.parameter=**pathname**	specifies the path and name of the AppMenu Parameter Specification File.
appmenu.defaults=**pathname**	specifies the path and name of the AppMenu file to retrieve and store parameter defaults.

appmenu.reportviewer=TedRE	AppMenu will run the report with the /view switch to invoke TedRE to view the resulting .LIS file (see Appendix M.6.2 "Viewing Report Output".
appmenu.apvcaption=1	Display the TASKID and TSKDESC fields as the caption for the AppMenu Parameter Verification dialog box
appmenu.currentScreen=**gray[|on|off]**	Determines if the current screen shows up as a menu choice (off is the default).

6.17.14 TRANS Built-in Colors

You may use the following names when specifying colors in TRANS Environment File entries and in the TRS instruction file.

COLOR Name	RGB Value
antiquewhite	250, 235, 215
aquamarine	127, 255, 212
azure	240, 255, 255
beige	245, 245, 220
black	0, 0, 0
blue	0, 0, 255
brown	165, 42, 42
cadetblue	95, 158, 160
chocolate	210, 105, 30
cornflowerblue	100, 149, 237
cyan	0, 255, 255
darkblue	0, 0, 128
darkcyan	0, 139, 139
darkgray	169, 169, 169
darkgreen	0, 100, 0
darkkhaki	189, 183, 107
darkred	139, 0, 0
darkseagreen	143, 188, 143
darkviolet	148, 0, 211
deeppink	255, 20, 147
deepskyblue	0, 178, 238
dimgray	105, 105, 105
forestgreen	34, 139, 34

COLOR Name	RGB Value
gold	255, 215, 0
gray	192, 192, 192
green	0, 255, 0
greenyellow	173, 235, 47
ivory	255, 255, 240
khaki	240, 230, 140
lavender	230, 230, 250
lawngreen	124, 252, 0
lightblue	173, 216, 230
lightcyan	224, 255, 255
lightgray	211, 211, 211
lightgreen	144, 238, 144
lightskyblue	135, 206, 250
lightyellow	255, 255, 224
limegreen	50, 205, 50
linen	250, 240, 230
magenta	255, 0, 255
maroon	176, 48, 96
mediumblue	0, 0, 205
midnightblue	25, 25, 112
mistirose	255, 228, 225
navy	0, 0, 128
oldlace	253, 245, 230
orange	255, 165, 0
palegreen	152, 251, 152
pink	255, 192, 203
red	255, 0, 0
rosybrown	188, 143, 143
royalblue	65, 105, 225
saddlebrown	139, 69, 19
salmon	250, 128, 114
sandybrown	244, 164, 96
seagreen	46, 139, 87
skyblue	135, 206, 235

COLOR Name	RGB Value
snow	255, 250, 250
steelblue	70, 130, 180
tan	210, 180, 140
turquoise	64, 224, 208
violet	238, 130, 238
violetred	208, 32, 144
white	255, 255, 255
yellow	255, 255, 0
yellowgreen	154, 205, 50

Chapter 7:AdmREPORT: Creating Reports

The reporting tool (AdmReport.EXE) is a comprehensive tool for preparing a wide variety of printed reports. REPORT contains the tools for preparing simple lists, multi-level hierarchical subtotals, form letters, bills and checks on preprinted forms, cross-tabulations, schedules, etc. REPORT contains facilities for record selection, sorting, computation, table lookup, record linkage between files, and complex procedural logic.

REPORT reads the report instruction file (REP), and produces a report according to the instructions found there. There are two modes of formatting usable in a report, namely "automatic format" mode, and "explicit format" mode.

Automatic format mode is appropriate for detail listings in columnar format and/or summary reports, and is often used for ad hoc reports that are written quickly to be used only once.

Explicit formatting is used when the person writing the report needs to specify the exact output format.

Reports can produce files which are themselves instruction files for other ADMINS commands.

7.1 Outline of a Report Instruction File (REP)

Most reports use only a small subset of the REPORT facilities. The following outline shows the skeleton of a generic report, and is meant to provide a first look at the REPORT statements and their order of usage[1] in a report instruction file. All report instruction files are required to have a file type of ".REP".

```
REPORT              report name

FILE                file being reported from

SINGLE              single space the printout
PAGE                control pagination
LENGTH              non-standard page length
WIDTH               non-standard page width
INDENT              non-standard indentation of printout
OUTPUT              set output device
LP                  set printout options
SCALE               scale decimal values
NRECS               number of records to read (for testing)
FORMAT              use automatic formatting of output

SORT                reorders records on new sort fields

CREATE,SELECT,KEY,LINK,TABLE,RECODE,EXECUTE

HEADING
...                 print layout for heading of each page
END

DEF                 display file definitions on-line
CREATE              create a virtual field
SELECT              select records for printing
KEY                 select record based on key fields
LINK                get fields from a record in another file
TABLE               table lookup for descriptors or values
RECODE              logical constructs for cross-tabulation
EXECUTE             call the RMO
```

1. CREATE, SELECT, KEY, LINK, TABLE, RECODE, and EXECUTE may appear before the HEADING. They usually appear after the HEADING (where they are explained). The placement of these statements can have a significant impact on both the report logic and the report speed. These issues are discussed in Section 7.13 “Processing Statements”.

```
DETAIL statement    print values for each detail record
                    using automatic columnar format
DETAIL section      print values for each detail record
...                 based on explicitly specified format
END

TOTAL key/sort      designate control break for: subtotalling
      EOF           grand totals
      [PAGE]        page totals
      n             total after n records

SUPPRESS           suppress subtotal printout on single detail
 RESET PAGE n        reset page counter after control break
 EJECT               eject page after control break printing
 EJECT BEFORE       eject page before control break printing
                     summaries
 EJECT n           eject page after control break if n or less
                     lines left on page

 CREATE              create a virtual field

 PREVIEW
 ...               print layout to preview subtotalled detail
 END

 SUMMARY
 ...                 print layout for each subtotal summary
 END
```

7.2 REPORT Statement

Each report may have a name. The REPORT statement, if included in the REP instruction file, is the first statement in a report and is used to name the report:

```
REPORT name
```

The REPORT statement is optional if there is only one report in the REP instruction file. However, a REP instruction file may contain several different named reports. If the REP instruction file includes several reports, then the REPORT statement is optional only for the first report in the REP instruction file. All subsequent reports within the REP instruction file require a REPORT statement.

```
*  PAYROLL.REP
*
REPORT REGISTER
...
END
*
REPORT EARNINGS
...
END
*
REPORT DEDUCTIONS
...
END
```

To run a report the user types "REPORT" to the system prompt. REPORT then prompts for a "REPORT FILE NAME:". The user types the file name of the report instruction file (REP-file-name) and the name of the particular report within the report instruction file which is to be run. For example:

```
$ report
 REPORT FILE NAME:payroll earnings
```

To run the first report in a REP instruction file, only the name of the REP instruction file needs to be typed. Unless a specific report is requested after the REP-file-name, REPORT runs the first report in the REP instruction file. For example:

```
$ report
REPORT FILE NAME:earnings
```

The REP-file-name and the report name may be included on the command line. For example:

```
$ report payroll earnings
```

REPORT can optionally print a row of asterisks to show its progress through the file by placing the qualifier "STAR" at the end of the REPORT command line:

```
$ report payroll earnings -star
```

if the REPORT contains a SORT statement (see Section 7.12 “SORT Statement”), REPORT prints two rows of asterisks: one for the SORT pass, and one for the report pass. The “star” qualifier on the REPORT command line is ignored unless REPORT is producing a report output file (e.g., it does nothing if OUTPUT KB is specified).

Note that AdmReport for Windows can optionally display a “progress bar” or “activity indicator” as described in Section 7.22 “The REPORT Environment File”

7.3 FILE Statement

Each report runs on an input file. The FILE statement is used to name the input file for the report. Records from other files may be linked into the report. The order of the records presented in the report follows the order of the records in the input file, unless a SORT statement is used to reorder the output (see Section 7.12 “SORT Statement”). Also, the input file may be an index file, as described in Section 7.13.4.1 “LINK Example”, or it may be an RMO, as described in Section 7.19 “EXECUTE Statement: RMO Processing”.

The name of the input file follows the name of the report (if there is a REPORT statement):

```
REPORT BENEFITS
FILE PAY.MAS
```

If the REPORT statement is omitted, the FILE statement is the first statement in the report.

7.4 HEADING Section

Each report usually has a heading. A heading consists of information printed on the top of **each** page of the report. The heading can contain literal text and/or data. The "layout" of the heading follows in the lines after "HEADING" until the "END" statement is encountered. For example:

```
HEADING
CE   PAYROLL BENEFIT REPORT
BL
```

```
     DEPT EMPL# NAME           SICK BALANCE VACATION BALANCE
BL
END
```

The heading layout consists of lines of text that will be printed on top of every report page. Generally speaking, the heading will be printed as it appears in the layout. Layout information is indented five (5) spaces in the REP instruction file. The first five spaces are left blank or used for one of the following instructions:

- CE - instructs REPORT to "center" the line on which the "CE" appears. All centering is performed with respect to the active WIDTH and INDENT values.[2]
- BL - instructs REPORT to insert a blank line at that point in the report.
- L/C - instructs REPORT to act as if the next line in the layout were at line "L" and column "C" in the heading. "L/C" appears on its own line.
- L%C - use the line/column method described above, but the next output line will not "float" up if any previous lines are suppressed because they are completely blank. That is, any line designated with "L%C" will ALWAYS appear at the line designated (unless it is completely blank).[3] This feature is especially useful on pre-printed forms.
- L/R
- L%R - use "R" (for "right justify") in place of a column number in the L/C or L%C syntax. to print the layout line flush with the right margin. If a right justified line ends with a field, the field itself should be right justified (e.g., ---FLD).

The previous example using the "L/C" notation would appear as follows:

```
HEADING
CE   PAYROLL BENEFIT REPORT
BL
     DEPT EMPL# NAME
3/31
     SICK BALANCE  VACATION BALANCE
BL
END
```

Multiple lines of literal text following a line and column designation ("L/C") are interpreted to be positioned at "L/C", "L+1/C", "L+2/C", etc. That is, the line number increments by one, and the column setting (or right justification) stays in effect until it is changed by another "L/C" or the END is encountered. Also, if a line and column designation is used, it must be reset to column 1 before a "CE" line may be used, as is illustrated in the next example.

2. The WIDTH statement is described in Section 7.17.4 “WIDTH Statement”, the INDENT statement is described in Section 7.17.6 “INDENT Statement”.
3. Note that the line following L%C will be suppressed if it is completely blank, i.e. subsequent non-blank lines (until the next L/C or L%C) float up to the line and column indicated by L%C.

Headings may include today's date (TODAY), current time (NOW), and the current page number (PGNO). These "internal" fields are described in Section 7.16 "Internal Field Names". The following example shows these fields being used. The significance of the dashes is described in Section 7.6 "DETAIL Section".

```
HEADING
1/50
     TODAY----  NOW-----  PAGE: PGNO--
2/1
CE   STATUS OF ACCOUNTS
...
END
```

The HEADING section can also print data from records in the file. That is, the HEADING layout can contain print field designators as described in Section 7.6 "DETAIL Section". The data printed in the heading is taken from the first detail record printed on the page containing the heading. Data fields should be placed in the HEADING only if there is a DETAIL statement or DETAIL section.

REPORT automatically prints column headings for the data fields to be displayed when using automatic formatting. (Automatic formatting is invoked by the presence of a FORMAT statement (see Section 7.17.12 "FORMAT Statement") or a DETAIL statement (see Section 7.5 "DETAIL Statement").) The column headings consist of either the field names from the DEF, or they are taken from a data file previously prepared for this purpose (see Section 7.20 "Data Description File for Automatic Formatting"). These automatically generated column headings appear below the user specified heading in the report output.[4]

7.5 DETAIL Statement

The DETAIL statement is used when REPORT is to perform automatic formatting of the report output. In contrast, the DETAIL section (see Section 7.6 "DETAIL Section") allows the user to precisely format fields and literal text in the report output. The DETAIL statement and the DETAIL section are mutually exclusive: only one of these two features may be used within a single report.

The DETAIL statement is a single line beginning with the word "DETAIL" followed by the field names and literal strings[5] in the order they are to appear, and (optionally) the print widths of the fields to be displayed.

The general syntax of the DETAIL statement is:

```
DETAIL [*keyword] fld1[/n] [literal] fld2[/n][literal] fld3[/n]...
```

Literals that have imbedded blanks must be enclosed in single quotes. One detail line is printed for each report record.

In the following example the fields FIRST LAST and CITY are to be printed, along with the some literal character strings:

```
DETAIL 'Mr./Ms.' FIRST LAST 'of' CITY 'U.S.A.'
```

4. The automatic columnar headings can be suppressed using the *NOHEAD keyword in the DETAIL statement. See Section 7.5 "DETAIL Statement".
5. Any character strings in the DETAIL statement that do not match field names (or abbreviations of field names) are printed as literals.

Note that character strings enclosed in quotes are always treated as literals.

In the DETAIL statement, field names may be abbreviated to the initial letters which unambiguously identify the field.

```
DETAIL EMP# FIRST LAST STR NU
```

The DETAIL statement above instructs report to print the fields EMP#, FIRST, LAST, STREET and NUMBER in that order from left to right on the detail line.

When the DETAIL statement is followed by an asterisk "*", all fields in the input file definition are placed on the detail line in the report output. The fields appear in the order found in the internal input file definition.

```
DETAIL *
```

Each of the fields listed in the DETAIL statement is printed in columnar format in the report output, underneath a column heading. Since the columns of field values must fit on one line, the display is limited to the output page width specified in the WIDTH statement (see Section 7.17.4 “WIDTH Statement”). Once that width has been reached any remaining field columns cannot be printed. REPORT displays a message that the width is not sufficient and that the layout will be truncated. Hence, the "*" option would only be used on data files with few fields.

The automatically generated column headings consist of either the field names from the DEF, or they are taken from a data file previously prepared for this purpose (see Section 7.20 “Data Description File for Automatic Formatting”). These headings appear below the user specified HEADING in the report output. The automatic generation of column headings is disabled using the NOHEAD keyword, as follows:

```
DETAIL *NOHEAD FIRST LAST CITY
```

The asterisk used to identify NOHEAD is a special keyword, and is required. "*NOHEAD" may be abbreviated to "*N".

Ordinarily if a field name appears more than once in the DETAIL statement, the later occurrences are ignored. If you want fields to be repeated in the DETAIL statement, you must include the REPEAT keyword, then repeat the field name in the desired sequence, as in the following example.

```
DETAIL *REPEAT CITY LAST FIRST CITY
```

As with NOHEAD, the asterisk is required, and "*REPEAT" may be abbreviated to "*R".

The DETAIL statement is limited to one line in the REP instruction file and is not followed by an END statement.

7.5.1 Specifying Field Widths in the DETAIL Statement

REPORT uses default display widths to determine the size of the columnar layout for each field. .

Field Type	Default Width
I - Integer	7
Ln - Longword decimal	9 + number of decimal places
Dn - Decimal	9 + number of decimal places
Fn - Four-word decimal	26
DA - Date	9
DT - Date	11
An - Alphanumeric	characters in field (n)
Xpic - Picture	characters in picture (pic)

These default field widths may be overridden by appending "/n" after the field name in the DETAIL statement where "n" is the desired field width in characters.

```
DETAIL field1/n field2 field3/n ...
```

In the DETAIL statement above, field1 and field3 widths are explicitly specified while field2 is displayed with its default width.

Detail Statement	Result
DETAIL EMP# FIRST LAST	EMP# FIRST LAST 1021 Audrey Allen
DETAIL EMP# FIRST/1 LAST	EMP# F LAST 1021 A Allen

In the second example above, the DETAIL statement provides EMP# (X9999) with four places, LAST (A16) with sixteen places, and overrides the default print width of FIRST (A16) by allowing only one place. Note that the column heading for FIRST is truncated to the specified print width of one character. Meaningful column headings and alternative print widths may be set up in a table file used by REPORT to perform automatic formatting. This is described in Section 7.20 "Data Description File for Automatic Formatting" below.

7.5.2 DETAIL *CSV: Output in CSV Format

REPORT supports direct output of CSV-format files[6] via the *CSV keyword in the DETAIL statement.

Syntax:

```
DETAIL *CSV[AUTO] *
```

or

```
DETAIL *CSV[AUTO]  fld1  [fld2…] [:]
```

To output all the fields in the REPORT virtual record[7] use

```
DETAIL *CSV *
```

To generate an automatic "header line" (see below) use

```
DETAIL *CSVAUTO *
```

All the elements that follow the *CSV keyword in the DETAIL statement must refer to fields previously identified in the REPORT virtual record[8]. No "literals" are permitted.

The DETAIL *CSV statement displays non-numeric data-type fields in wrapped in quotes. Trailing blanks are discarded.

For example (continuation lines are permitted):

```
DETAIL *CSV block lot unit acct 1addr 2addr :
       value sdate sprice
```

Here is an excerpt from the output produced by this statement:

```
"100","24","","106050","176 PLEASANT STREET","",182000,"30-Jul-1980",.00
"100","25","","292100","SEYMOUR SLIVE & ZOYA S. SLIVE","174 PLEASANT ST",200200,"23-Sep-1988",.00
"100","26","","366050","172 PLEASANT ST","",278500,"17-Jul-1978",.00
"100","29","","491850","CORPORATION, THE","C/O FIRST ESTATE REALTY",315000,"25-Jul-1989",.00
"100","30","1","996016","TRUSTEE OF XERIC TRUST","31 SHEPARD ST",139300,"23-Sep-1997",.00
"100","30","2","996017","TRUSTEE OF XERIC TRUST","31 SHEPARD ST",139300,"23-Sep-1997",.00
"100","30","3","996018","TRUSTEE OF XERIC TRUST","31 SHEPARD ST",154700,"23-Sep-1997",.00
"100","30","4","996019","222-224 CHESTNUT ST. UNIT #4","",170200,"23-Sep-1997",180000.00
"100","30","5","996020","222-224 CHESTNUT ST. UNIT #5","",170200,"29-Sep-1997",190000.00
```

Often applications that import CSV files can use field-identifying information ("header" information) if it is placed on the first line of the output in comma-separated format. REPORT allows you to specify the CSV format header line directly in the .REP instruction file, or generates one automatically on request.

Substituting *CSVAUTO for *CSV in the detail statement tells REPORT to automatically generate a header line.

```
DETAIL *CSVAUTO block lot unit acct 1addr 2addr :
       value sdate sprice
```

---

6. CSV or "comma separated values" format files are ASCII text files with one record per output line, and data values separated by commas. CSV-format files provide easy data exchange with spreadsheet applications such as Microsoft Excel, database applications such as Microsoft Access, and 3rd party reporting applications such as Crystal Reports.
7. A maximum of 250 fields is allowed.
8. Field names may be abbreviated to the initial letters, which unambiguously identify the field.

The automatically generated header line contains the labeling text provided for each field in the Data Dictionary. If no labeling text for a field is found in the data dictionary (or if the Data Dictionary is not active) the field name is used.

The following example shows how to specify a "hard-coded" CSV-format header line directly in the .REP:

```
file gisreal.mas
NRECS 100
LP 0 1 0 - doc.csv
WIDTH 250
HEADING
     Block,Lot,Unit,"Account Number",Address1,Address2,
     "Assessed Value","Sale Date","Sale Price"
END
DETAIL *CSV block lot unit acct 1addr 2addr value sdate sprice
```

Note in this example that the HEADING section contains multiple lines. HEADING lines are handled in a special way when DETAIL *CSV is in use: they are concatenated together to produce a single output line[9].

By default AdmReport uses “,” (comma) as the field separator when producing CSV output[10]. Many applications (especially outside the US) have special requirements, so any character can be specified as the field separator via the logical name ADM$CSVSEPARATOR. You may assign any printable character except the numbers 0-9 to this logical name, or assign the decimal ASCII code for the character.

For example:

```
AdmLcr ADM$CSVSEPARATOR ;
```

and

```
AdmLcr ADM$CSVSEPARATOR  59
```

will both use “;” (semicolon) as value separator. And

```
AdmLcr ADM$CSVSEPARATOR 9
```

will use TAB as separator character.

9.Note that when the REPORT instruction file is read, the HEADING section is encountered before the DETAIL *CSV statement. If the HEADING line extends beyond the WIDTH (default 131) of the report, REPORT exits with the diagnostic message

```
rep976 Print page width error
```

To avoid this use the "-CSV" command line option (see Section 7.5.2.1 “Embed CSV syntax in "multi-purpose" report”) so that REPORT "knows" its outputting CSV format when the HEADING is encountered, or increase the WIDTH.

10. If Option K (see Appendix A: “Options”) is active a TAB character is used as the delimiter.

There are some other "special behaviors" in effect when DETAIL *CSV is in use.

Special Behavior	Explanation
Limits	Ordinarily DETAIL statements or "automatically formatted reports" are limited to a maximum of 50 fields. When DETAIL *CSV is in use you may reference up to 250 fields. Ordinarily REPORT output lines are limited to 254 characters long. When DETAIL *CSV is in use output lines may be any length.
REPORT instructions	DETAIL *CSV report output is always one line per record output to a file. Formatting, (e.g. WIDTH, LENGTH, etc.) and TOTAL statements are ignored. Continuation lines are permitted (for DETAIL *CSV line)
HEADING	Layout lines concatenated
-CSV switch	*!CSV! token enables multi-purpose reports (see next section)
Option "P"	Ignored (negative values are always displayed with a preceding "-", e.g. -43.75)
Option "K"	Decimal point in numeric field is a "comma", fields are separated by a tab character.

7.5.2.1 Embed CSV syntax in "multi-purpose" report

Build multi-purpose REPORT instruction files by using the "***!CSV!**" token in the .REP file and the "**-CSV**" REPORT command line option. The *!CSV! token identifies lines in a report that are to be read only when the -CSV command line option is invoked. This option allows developers to assemble a REPORT virtual record and output the data in either traditional or CSV format.

Command line syntax is:

```
ADMREPORT -CSV [=file] reportname
```

For example:

```
ADMREPORT -CSV BUDGET
```

produces an automatically named output file, e.g. "admins000111a.lis", or

```
ADMREPORT -CSV=MYFILES:BUD.CSV  BUDGET
```

!to specify the name of the output file (BUD.CSV)

Consider the following .REP file:

```
*  FEDEDUCT.REP
*
FILE DEDUCT.MAS
SINGLE
WIDTH 80
HEADING
1/1
     TODAY------
2/1
CE   STATE AND FEDERAL DEDUCTION CHECK REPORT
END
LINK NAME ADDR CITYST ZIP DESC 2DESC FROM NAMES.MAS KEY IS CODE
LINK 3DESC 4DESC 5DESC FROM NAMES.MAS KEY IS CODE
CREATE XFICA/D2 FICA + CFICA
CREATE XMED/D2 MED + CMED
```

```
*
*!CSV!DETAIL *CSVAUTO TODAY NAME ADDR CITYST ZIP AMT :
*!CSV!        WITH XFICA XMED DESC
DETAIL
BL
BL
     CKDATE-----
       NAME---------------------------------- ----------AMT
       ADDR----------------------------------
       CITYST---------------------------------
       ZIP--
BL
       DESC-----------------------------------------------
BL
1     ( WITHHOLDING: ------WITH )
2     (         FICA: -----XFICA )
3     (    MEDICARE: ------XMED )
BL
           --------------------
END
```

When run conventionally this report will produce conventional output: an (automatically named) .LIS laid out as specified in the .REP file's DETAIL section. REPORT will ignore the lines beginning with *!CSV! because they are marked as comments. However, when this instruction file is run with the -CSV command line option, as follows:

```
> ADMREPORT -CSV=MYFILES:FED.CSV FEDEDUCT
```

REPORT searches for and reads the lines that begin with the *!CSV! token. These lines are processed (with the *!CSV! token removed) and conventional REPORT layout and formatting instructions are ignored. REPORT -CSV effectively "sees" the following instruction file:

```
FILE DEDUCT.MAS
LINK NAME ADDR CITYST ZIP DESC 2DESC FROM NAMES.MAS KEY IS CODE
LINK 3DESC 4DESC 5DESC FROM NAMES.MAS KEY IS CODE
CREATE XFICA/D2 FICA + CFICA
CREATE XMED/D2 MED + CMED
DETAIL *CSVAUTO TODAY NAME ADDR CITYST ZIP AMT :
        WITH XFICA XMED DESC
```

7.6 DETAIL Section

The DETAIL section is used instead of the DETAIL statement when one needs to specify exactly how the values in the report records are to be printed. The DETAIL section allows the user to specify the layout of the fields for each record in the input file. Unlike the DETAIL statement, the DETAIL section allows multiple lines of detail output for each record and the placement of literal text among the detail fields.

The DETAIL section, which begins with a line consisting of the word "DETAIL" and ends with the "END" statement, instructs REPORT what to print for each record in the report input file. The layout in the DETAIL section looks similar to the HEADING section, i.e., it is indented five (5) spaces and it can contain "CE", "BL", "L/C", and literal text. Of course, the DETAIL section can also print data from the records in the report input file. The user instructs REPORT what data is to be printed, and where it is to appear on the output page, by typing the data field name with left-leading or right-trailing dashes in the layout section. The data field name together with the

dashes is called a "print field designator". The print field designator is positioned in the DETAIL layout at the place where the data is to be printed in the report. There are several rules associated with the convention for the print field designator:

1. The number of characters printed for each field is equal to the total number of characters (data field name plus dashes) in the print field designator.
2. The data field name need **not** be complete. Rather the data field name should contain sufficient initial letters from the full primary field name (as it appears in the DEF) to distinguish this data field name from all other primary names in the DEF. Provision of sufficient letters is the user's responsibility. REPORT will take the **first** data field name from the DEF which it finds that matches the print field designator. The order of the search follows the order of the names in the DEF.

   A problem arises when there are two field names in a DEF where one is included as a substring of the other, e.g. if the field "ST" follows the field "STREET" in the DEF. When specifying a print field designator one can add a period (".") to a field name to indicate that one is specifying the **full** field name, and not a partial name. In our example, "ST" is ambiguous as to whether the reference is to "ST" or "STREET", whereas "ST." is specifically a reference to "ST" and "STR" is a reference to "STREET".
3. Left or right justification in the printing field is controlled by the placement of the dashes. Left dashes means right justification. Right dashes means left justification. For example, "---SICKB" right justifies the sick balance value whereas "SICKB---" left justifies the sick balance value. Dashes should appear either to the right or to the left of the data field name. A string containing both a beginning and an ending dash or dollar sign (see below) will be treated as a literal. Hence a string of all dashes is a literal.
4. The "$" may be used interchangeably with the "-". For example, by using "$$$$$GROSS" instead of "-----GROSS", a **single** dollar sign will precede the value and be printed flush against the value.
5. The actual value is placed in the designated print field during report generation. If the actual value is larger than the designated field, it will be truncated from the right. If the actual value is smaller than the print field, blanks will fill out the field either in the leading or trailing position depending on the justification.
6. All numeric data is automatically edited for commas and decimal point where appropriate.

DETAIL section example:

```
DETAIL
     D--  EMP-- NAME---------------
1/31
       ---SICKB         ---VACB
END
```

The discussion between here and Section 7.6.1 "Text Fields" is for the more advanced readers of this manual, and can be skipped by the beginning reader.

Using the field names and the dashes you can easily visualize how the report is going to look. However, there are times when because of the length of the field name needed to uniquely identify the field or because we wish to fully utilize a print line, the "line/column" technique is required. Using the "line/column" technique, the print line may be defined a character at a time if necessary. Also, blanks that are between print field designators do not erase data characters already placed in the line. The following examples expand on the use of the "line/column" technique.

```
HEADING
      ACCOUNT DESCRIPTION
END
DETAIL
```

```
     FUND-
1/3
     DEPT-
1/6
     OBJ-
1/1
               DESC--------------
END
```

Note that in the above example, three fields (FUND, DEPT, and OBJ) are packed together without any blanks. Also, note that the column was "reset" back to 1 for the DESC field so that it could be lined up visually with the column heading (DESCRIPTION). The blanks in the line leading up to DESC--- do not erase the account number already formatted. For FUND 01, DEPT 514, and OBJ 110 the above example would print:

```
 ACCOUNT DESCRIPTION
01514110 BROOMSTICK HANDLES
```

If the line/column designation is absent, printing will continue on the next line but at the same column position contained in the previous line/column designation. So that:

```
2/27
     -----AMT1
3/27
     -----AMT2
4/27
     -----AMT3
```

is more succinctly written:

```
2/27
     -----AMT1
     -----AMT2
     -----AMT3
```

7.6.1 Text Fields

Blocks of text from a document stored in or associated with a text field[11] (TInn or TXnn field type) can be integrated into REPORT outputs. The block of text can be either the entire document, or limited to a specific maximum number of lines, using the following syntax:

```
TEXTFIELD----------height---
```

For example, to print up to 6 lines of text stored in the TI field ADDRESS:

```
ADDRESS-------------6---------------------------------
```

The height can be anywhere within the dashes of the field designator (but it must be surrounded by dashes). Since the maximum length of a section (DETAIL, SUMMARY, etc.) is 63 lines, the number of lines to be output in the text block, plus the line number within the section where the text block begins, minus 1, cannot exceed 63.

If the height specified is 0:

```
FIELD------------0---
```

REPORT prints the entire text regardless of its length (unconstrained by the 63 line limit described above). If this syntax is used, **nothing else can be placed in the same columns as the text field on lines below it in the layout section,** because this might

11. See Appendix K: “Using Text Fields”.

result in overprinting (see below). If this open-ended text block syntax causes page breaks, the HEADING will be correctly paginated (i.e. the PGNO is correctly maintained); but other data in the HEADING may be late. Except for PGNO, HEADING sections caused by open-ended text block page breaks are simply copies of the HEADING section on the page where the text field began.

Text blocks are always left justified. "CE" (center) is not supported for blocks of text.

You can use text fields in LINK, TABLE, and TOTAL statements. You can use text fields with automatic link field renaming. You cannot use text fields as SORT fields.

In TOTAL statements, the aggregation operators /FI, /LA, and /E are available for TI and TX fields; others result in an error message. Each text field in the file, in a LINK, TABLE, and/or a TOTAL with /FI or /LA counts as **TWO fields**[12] against REPORT's limit of 1000 fields.

Text block formatting uses the "L" or "J" justification code in the ruler stored with the text, and uses the print width specified in the layout. "L" lines (no justification) are truncated if they are too long.[13]

If, when you lay out a text block, you specify a maximum number of lines to be displayed which is greater than the number of lines in the document, the remaining lines of the text block are "displayed" as blank. If this results in a completely blank line, it is suppressed (as always in REPORT). But if the text contains embedded blank lines, they are not suppressed.

When laying out fields which are directly underneath a text block, be sure to allow enough lines under the text block to prevent overprinting. For example:

```
WRONG: may overprint              RIGHT: won't overprint
====================              ======================
1/1                               1/1
      TXT1----------4--                 TXT1----------4--
4/1                               5/1
         FIELD-------                      FIELD-------
```

REPORT detects overprinting in text fields and issues an error message if it occurs.

12. Each text field has an internal field (TI$field_name of TX$field_name) associated with it, as described in Appendix K: "Using Text Fields".
13. Rulers are explained in J.3 "Rulers".

7.6.1.1 Substituting Values into Text Fields at Run Time

Values from the current virtual record can be merged into a block of text at run time by referencing the field in the text document as follows:

```
<%%FIELDNAME>
```

JANBUD, a field in REPORT's virtual record, would be substituted into the text block when the text field that contains the following lines is printed using REPORT:

```
The total budget for January is $<%%JANBUD>, we will be
considering at our next meeting how to allocate these
funds in an equitable manner. We hope you can be there.
```

If JANBUD contained the value $2,408.50, the REPORT outputs the lines from the document as follows:

```
The total budget for January is $2,408.50, we will be
considering at our next meeting how to allocate these
funds in an equitable manner. We hope you can be there.
```

7.6.2 Zero Suppression

Any print line may contain zero suppression. This means that zero values for all numeric, date, time, and picture fields print as blanks. To instruct zero suppression place a "Z" in column 1 of the line containing the print field designators where zeroes are to be suppressed. For example:

```
DETAIL
Z      -----ENCUMB  -----EXPEN  -----BAL
END
```

7.6.3 Comma Suppression

Commas may be suppressed when decimal or integer values are being printed in a report. The letter "C" is inserted in the first column of the line containing the print field designators in which commas are to be suppressed. Zero and comma suppression may be requested together. For example:

```
DETAIL
1/40
C    ------AMT
ZC   ------BAL
END
```

The AMT field will have its commas suppressed. The BAL field will have its commas suppressed and a value of zero will print as blanks.

7.6.4 Explicit Print Field Designator

In certain cases the use of dashes in the print field designator is unwieldy. For example, if the printout field is only one or two spaces wide. A "D" in column 1 of a line containing print field designators instructs REPORT that each string on this line is a print field designator, even though there may be no dashes present in the string. For example:

```
DETAIL
D      FU     DEP     OBJ    SERV--
END
```

7.6.5 Explicit Print Field Width

Notwithstanding the "D" feature just described, the assignment of field width via the field name in the print designator may still in certain cases be unwieldy. For example, a one character print field is required for a field name whose initial letter leaves the identity of the field ambiguous. When REPORT sees "Dn", as in "D1", on a line containing print field designators this instructs REPORT that the string(s) on this line are field names that should be printed in print fields of width "n". An example of this could be where only part of a field was to be printed. For example, if we had several code fields, each of the type "A4", and we only wanted to print the first character of each field, the following could be used:

```
DETAIL
D1     S1 S2 S3 S4 S5 S6 S7 S8
END
```

7.6.6 DETAIL Subheadings

In some reports, it is desirable to print some additional information at the beginning of each control break or at the beginning of each page, on the same line or lines as some of the DETAIL information, as follows:

```
KEY = 1  DETAIL ---------------------------------
         DETAIL ---------------------------------
         DETAIL ---------------------------------

====== SUMMARY for KEY = 1 ======

KEY = 2  DETAIL ---------------------------------
         DETAIL ---------------------------------
         DETAIL ---------------------------------

               ******************
               **** HEADING ******
               ******************

KEY = 2  DETAIL ---------------------------------
         DETAIL ---------------------------------

====== SUMMARY for KEY = 2 ======

KEY = 3  DETAIL ---------------------------------
         DETAIL ---------------------------------
```

In this example, "KEY = n" is printed on the first DETAIL line after each control break, and on the first DETAIL line on each page.

To provide this capability REPORT has two options for use in DETAIL paragraphs. These options are controlled by two codes in the left margin. 'P' tells REPORT to print the contents if the layout line only for the first record on each page.[14] 'F' tells REPORT

14. The 'P' code to print a detail subheading for the first detail section on each page should not be used in combination with multiple output files (the DIRECT statement, see Section 7.17.14 "DIRECT Statement: Multiple Output Files"). REPORT determines when to print 'P' lines according to the page breaks in the original report; so, unless the final output files containing DETAIL sections have page breaks before the same detail records, the 'P' subheadings will not be placed correctly in the final output.

to print the contents of the layout line only for the first record in each control break. 'P' and 'F' can be combined, as in the example above, which would be produced using the following layout:

```
HEADING
                        ******************
                        **** HEADING ******
                        ******************
BL
END
DETAIL
PF      KEY = K-
1/1
                  DETAIL ----------------------------------
END
TOTAL KEY
SUMMARY
BL
        ====== SUMMARY for KEY = K- ======
BL
END
```

Subheading information can appear on any line or lines within a multi-line DETAIL section. 'F' and 'P' can be used in combination with the 'C' and 'Z' codes, and these codes can be given in any order. 'F' and/or 'P' must appear in the left margin of EACH LAYOUT LINE which is to be controlled by these options.

The 'F' subheading code can be followed by a number between 1 and 10, indicating that the line should be printed following each control break for the given total. For example:

```
DETAIL
PF2   KEY1-
1/1
PF1         KEY2-
1/1
                 Detail line --------------------------
END
TOTAL KEY2
...
TOTAL KEY1
```

In this example, KEY1 is printed whenever there is a control break on the second TOTAL or there is a page break; and KEY2 is printed at each control break on the first TOTAL and each page break. Since a control break on the second TOTAL always causes a control break on the first one, this produces output such as:

```
Key_1 Key_2 Detail
----- ----- ----------------------------------------
  1     1   One, one
        2   One, two
  2     1   Two, one
  3     2   Three, two
            Another three, two
        4   Three four
   4     1   Four, one
```

If there is no total number after the 'F' code, REPORT defaults to the first total.

7.7 TOTAL Statement

REPORT can perform up to ten (10) levels of subtotaling, unless RECODE or SUPPRESS is used, in which case REPORT can perform up to six levels of subtotaling. The TOTAL statement specifies the control break, and which subtotaling operations are to be applied to which fields. At each TOTAL break REPORT can sum values, count values, take an average, find the largest or smallest value within the run of the subtotal, or take the first, second, third, fourth, or last value in the run. Many fields can be subtotaled, and several subtotaling operations can be applied to the same field.

The general syntax of the TOTAL statement is:

```
TOTAL control  field1[/operation] field2[/operation] etc.
```

The "control" is a key field or sort field from the file or a derived key from a SORT statement (see Section 7.12 “SORT Statement”), or one of the control break options described below. The "control" instructs REPORT when to break and perform subtotaling operations. For example, when the control break is a field name (e.g. a key field) REPORT will break each time the value of that key field or a higher key field changes.

The TOTAL statement may be continued on another line using the colon (:) continuation syntax only when explicit formatting mode is active.

The subtotaling operations that can be performed on fields are /V, /E, /AVG, /MAX, /MIN, /FI, /2, /3, /4, and /LA which stand respectively for (sum) values, (non-null) existences, average, maximum, minimum, first, second, third, fourth, and last. (The purpose of the "/n" operations are illustrated in Section 7.10.2 “Multi Column Reports”.)

Summing values (/V) is the default operation for numeric fields if no operation code is present. For all other data types the first occurrence in the control break (/FI) is the default operation. If the maximum value for an integer field (i.e. 32,767) is reached by summing the values for that field, a warning message displays and the field is set to 0. This condition does not stop the report from running to completion.

The TOTAL statement may be used with either automatic formatting or explicitly specified formatting of the report output. The use of the TOTAL statement with automatic formatting is described in Section 7.8 “Subtotaling with Automatic Formatting and DETAIL” and Section 7.9 “Subtotaling with Automatic Formatting without DETAIL”. SUMMARY and PREVIEW sections which allow the user to explicitly specify the layouts of subtotaled data are described in Section 7.10 “SUMMARY Section” and Section 7.11 “PREVIEW Section”. SUMMARY and PREVIEW sections may not be used when automatic formatting is invoked.

Both actual fields and fields derived at the detail level (i.e., via CREATE, LINK, TABLE, and RECODE described below) may be totaled at any break. However, fields derived at one control break **may not** be totaled at another break.

There are various options possible for the "control" break. These options can be used when either automatic formatting or explicit formatting is selected. These "control" break options are as follows:

7.7.1 Break At A Key or Sort Field Change

The "control" may be a key field (or a sort field, see Section 2.4.3 “Sort and Access Control”) of the file named in the FILE statement, or a key derived by the SORT statement (see Section 7.12 “SORT Statement”). Fields that are included in the TOTAL statement cannot precede the control field in the sequence of key fields from the DEF or the SORT statement. For example, if the break was on the KEY3 field then the KEY2 field could not be summed in that TOTAL statement.

For example:

```
TOTAL DEPT SICKB VACB
```

The above example subtotals sick and vacation balance for each department.

```
TOTAL DEPT SICKB SICKB/E SICKB/MAX SICKB/MIN SICKB/AVG
```

The above example subtotals sick balance per department, counts the non-null existences ("E") of sick balances per department, finds the largest ("MAX") and the smallest ("MIN") sick balance per department, and takes the average ("AVG") sick balance per department.

TOTAL sections must appear in order from minor to major key/sort fields.

There may be more than one TOTAL statement using the same key or sort control field. This is useful in the case where a control break might only include a single detail record. If the user desired to only print the total when there was more than one record, but wanted a blank line at each break, the following technique could be applied. (SUMMARY and SUPPRESS are presented completely in Section 7.10 “SUMMARY Section” and Section 7.18.1 “SUPPRESS Statement”)

```
TOTAL OBJ AMT
SUPPRESS
SUMMARY
1/50
     --------
     -----AMT
END
TOTAL OBJ
SUMMARY
BL
END
```

7.7.2 Break At End of File

The "control" may be "EOF". This means at the end of file produce the subtotaling operations based on the whole file, i.e. produce grand totals. For example:

```
TOTAL EOF SICKB VACB
```

would produce grand totals of sick balance and vacation balance.

7.7.3 Break At End of Page

The "control" may be "[PAGE]". This means print page totals at the end of each page. The "[PAGE]" control break supersedes other key/sort field control breaks. Also "[PAGE]" is only meaningful if DETAIL is also active in the report. For example:

```
TOTAL [PAGE] SICKB VACB
```

would produce page totals for sick and vacation balance. Generally "[PAGE]" is not used with breaks on sort fields; "[PAGE]" is usually used with DETAIL and TOTAL EOF.

7.7.4 Break At A Fixed Number of Records

The "control" may be a number, "n". This means take a control break after each n records have been printed. This feature may be used to produce multi-column reports (illustrated in Section 7.10.2 "Multi Column Reports") or to produce batch subtotals for fixed-length batches.

7.7.5 Break At Partial Field

The "control" may be of form "sort-field partial". Partial is a code for taking a subtotal when only **part** of the control field changes. The codes for partial follow, and depend on the data type for the control field. The partial code is prefixed with an "=" to instruct REPORT that it is a partial.

1. Control field is a date. Partial can be "=YY" or "=YY-MMM" to subtotal by year or month. The date should be in the standard ADMINS date format, DD-MMM-YY or DDMMMYYYY. For example:

   ```
   TOTAL INVDATE =YY-MMM ...
   TOTAL AGE =YY ...
   ```

2. Control field is "pictured". Partial is the leading part of the picture showing the part of the data that should change to produce the subtotal. For example, "=9" or "=AA9" are partials for full pictures of "999" or "AA9999".

   ```
   TOTAL FUND =9 ...
   TOTAL CODE =AA9 ...
   ```

3. Control field is alphanumeric, i.e. An. Partial code is of form "=XXXX" where the number of X's in the code controls the number of alphanumeric characters, from the left, that make up the partial field whose change activates the subtotal.

   ```
   TOTAL NAME =X HOUSE LOT OUTBLDG STORE :
     FACTORY ACRES :
     VETERAN BLIND DISABLD ELDERLY
   ```

4. When the control field is has sub-fields the partial key can be a subfield. For example, if ACCOUNT.DEPT is a subfield of ACCOUNT:

   ```
   TOTAL ACCOUNT =.DEPT...
   ```

7.8 Subtotaling with Automatic Formatting and DETAIL

When a DETAIL statement is present in the REP instruction file, the TOTAL statement uses the columnar layout derived from the DETAIL statement. The SUMMARY section is not allowed.

The minimum requirement for the automatic formatting of totals is a designated control break. All decimal and four-word decimal fields from the DETAIL statement are automatically summed and printed at each TOTAL break, without having to include these field names in the TOTAL statement. The user may override the automatic subtotaling of these fields as described in Section 7.20 "Data Description File for Automatic Formatting" below.

```
TOTAL control
```

Specific aggregation operators (e.g. /MIN, /LA) may be requested as described in Section 7.7 "TOTAL Statement" on the TOTAL statement.

```
TOTAL control field1/operation field2/operation etc.
```

When automatic formatting is active, REPORT automatically applies the "/V" operation to numeric fields (i.e. decimal and four-word decimal fields), and the "/FI" operation to all non-numeric fields, included on the TOTAL statement. Decimal and four-word decimal fields do not have to be included on the TOTAL statement to be automatically subtotaled. Integer fields, which are not automatically totaled by default, may be explicitly requested by including them on the TOTAL statement.

When a numeric field is summed, the result is placed below a line of dashes, in the column containing detail values for that field. If any other field operation is requested (e.g. /MAX), REPORT automatically generates a separate additional column for each aggregated field value. The column heading indicates the field and the operation, and the results are placed on the total output line.

The print width of fields included in the TOTAL statement may be explicitly specified. To override the default display width of aggregated fields, append "/n" after the field/operation instruction where "n" is the desired width in characters. ("N" must be greater than 4, so as not to be confused with the "/2", "/3" or "/4" operation described in Section 7.10.2 "Multi Column Reports".)

```
TOTAL control field1/operator/n field2/operator/n ...
```

Field names in TOTAL statements must be fully spelled out. The TOTAL statement is restricted to one line (i.e. colon continuation is not supported) when automatic formatting is active. REPORT will automatically format a maximum of 50 fields, including DETAIL fields and TOTAL fields.

7.8.1 Automatic Formatting Examples, DETAIL and Subtotals

In the following example, the field SALARY (D2) is automatically subtotaled with a "control" break of end of file (EOF).

```
DETAIL DEPT EMP# LAST SALARY
TOTAL EOF

DE EMP# LAST                    SALARY
12 1021 Allen                23,000.00
12 1254 Cosmos               34,500.00
```

```
...                                  ...
                                -----------
                                  277,000.00
```

In the next example, the print width for the field LAST is set to 10 characters in the DETAIL statement. The report subtotals each department, displays the first occurrence of the department name (DNAME) for that control break, and automatically subtotals the SALARY field.

```
DETAIL DEPT EMP# LAST/10 SALARY
TOTAL DEPT DNAME/FI


DE DNAME/FI        EMP# LAST            SALARY
12                 1021 Allen        23,000.00
12                 1254 Cosmos       34,500.00
12                 1256 Short        27,000.00
                                    ----------
12 Personnel                         84,500.00
...
```

In the next example, the department name (DNAME) is truncated to 9 characters. The report prints a department subtotal for SALARY, and at the end of the file, a grand total for salary plus the maximum SALARY.

```
DETAIL DEPT EMP# LAST/10 SALARY
TOTAL DEPT DNAME/FI/9
TOTAL EOF SALARY/MAX


DE           EMP# LAST            SALARY  SALARY/MAX
12           1021 Allen        23,000.00
12           1254 Cosmos       34,500.00
12           1256 Short        27,000.00
                               ----------
12 Personnel                    84,500.00
...
                              -----------
                                  277,000.00  34,500.00
```

7.9 Subtotaling with Automatic Formatting without DETAIL

If a report which does not contain a DETAIL statement is to automatically format subtotaled values, then the FORMAT statement (see Section 7.17.12 "FORMAT Statement") preceding the TOTAL statement is necessary to invoke automatic formatting.

Since there is no information from a DETAIL statement as to print widths, the TOTAL statement uses the default widths, or the user specified overrides on the TOTAL fields, or the widths found in the ADM$FORMAT file as described in Section 7.20 "Data Description File for Automatic Formatting".

7.10 SUMMARY Section

The SUMMARY section allows the user to specify the layout of fields and literal text for subtotaled data derived by the TOTAL statement. SUMMARY precedes the layout for the subtotal printout. For each TOTAL statement, i.e. control break, there may be a SUMMARY "paragraph" of printout. The format of the SUMMARY printout may be the same as, or similar to, the format used in the DETAIL section printout, or may be formatted entirely differently from the DETAIL section printout. Also, a particular report may contain **only** SUMMARY section printout, and no detail section printout.

The layout of the SUMMARY is formatted using the same rules as in the HEADING and DETAIL layouts. That is, one uses "CE", "BL", "L/C", print field designators, and literal text. However, only the following fields are usable in the SUMMARY paragraph to make a print field designator in a SUMMARY layout.

1. Any field in the line of sort up to and including the control field in the TOTAL statement for this SUMMARY paragraph. For example, if the control field is the KEY2 field, then the KEY1 and KEY2 fields may be used. In order to use the KEY3 field it must be placed on the TOTAL statement with a subtotaling operation.
2. Any field used in the TOTAL statement for this SUMMARY paragraph with a subtotal operation.
3. Any field introduced by a CREATE statement that occurs between the TOTAL statement and the SUMMARY paragraph.[15]

For example:

```
TOTAL DEPT SICKB VACB DEPTNA/FI
SUMMARY
BL
     DE--  DEPTNA/FI---   TOTAL SICK BAL: -----SICKB
2/80
     TOTAL VACATION BAL: ------VACB
END
```

Note the use of the "/FI" operator to bring DEPTNA (department name), which may have itself been brought into the report via a TABLE statement, into the scope of the SUMMARY paragraph via the TOTAL statement.

Fields derived at one control break may not be totaled at another break.

15. Link and table fields, discussed in Section 7.13.4 “LINK Statement” and Section 7.13.5 “TABLE Statement”, should be introduced at the detail level and brought into a SUMMARY paragraph via the "/FI" operation.

7.10.1 Positioning The SUMMARY Section

REPORT can start the SUMMARY printout at a specified line. This feature is especially useful when output is to a preprinted form. For example, to print a total at line 40 when printing onto purchase order forms, the line number where the SUMMARY paragraph is to start is placed in the SUMMARY statement.

```
TOTAL VENDOR ITEMS AMOUNT
SUMMARY 40
     TOTAL FOR ORDER:  -----IT ITEMS, $$$$AMOUNT
END
```

7.10.2 Multi Column Reports

Sometimes the need arises to print multi-column reports. For example, consider the following file definition:

```
*   LIST.DEF
MAS 10000
NAME A20 KEY1     "name"
ADDR A20          "address"
CITY A20          "city, state, zip"
```

We wish to print mailing labels, two to a row on the printed output page. This could be done as follows:

```
SUMMARY
     NAME/FI------------     NAME/2--------------
     ADDR/FI------------     ADDR/2--------------
     CITY/FI------------     CITY/2--------------
END     REPORT LABEL
FILE LIST.MAS
HEADING
BL
END
TOTAL 2 NAME/FI ADDR/FI CITY/FI NAME/2 ADDR/2 CITY/2
```

"TOTAL 2" means take a control break every 2 records.

Three additional operators are available with TOTAL to facilitate printing 2, 3, or 4 column reports. These operators, "/2", "/3", "/4" respectively, are used to extract a value from the 2nd, 3rd, or 4th records preceding the control break. For example, to print the names from the mailing list in four columns, the following report could be run. (The report could easily be extended to print the full label.

```
REPORT NAMES
FILE LIST.MAS
HEADING
CE   LIST OF NAMES
END
TOTAL 4 NAME/FI NAME/2 NAME/3 NAME/4
SUMMARY
     NAME/FI------ NAME/2------ NAME/3------ NAME/4------
END
```

In these last two examples we again saw the technique of using a TOTAL operation (e.g. /FI, /2, /3, /4) to bring detail data "into" the scope of the SUMMARY statement, in this case to mix and spread the contents of several records evenly across report lines.

The TOTAL n facility should only be used in reports with one level of totaling. Do not use this multi-column summary syntax in conjunction with other TOTAL breaks.

7.10.3 Summary *CSV statement : CSV output based on TOTALs

Data aggregated by a TOTAL statement can be output in CSV format by using the SUMMARY *CSV statement instead of a SUMMARY paragraph.

The general syntax is:

```
SUMMARY *CSV[AUTO] field1 [field2 …]
```

or

```
SUMMARY *CSV[AUTO] *
```
[16]

Fields specified on a TOTAL statement are known within the range of the TOTAL statements as FIELDNAME/AGGOP, where AGGOP is one of the REPORT aggregation operators V, E, AVG, MAX, MIN, FI, LA, 2, 3, 4. The default aggregation operator for a field depends on the field type. For numeric fields (I, L, D, F) it is /V, for non-numeric data fields it is /FI.

When you specify fields on the SUMMARY *CSV it is a good practice to include aggregation operator with the field name, e.g. AMOUNT/V rather than just AMOUNT, because if you e.g. have

```
TOTAL EOF AMOUNT/MAX AMOUNT
SUMMARY *CSV AMOUNT
```

you will get the first occurrence of AMOUNT on the TOTAL line, which is AMOUNT/MAX and not AMOUNT/V as might be expected.

If you use the SUMMARY *CSVAUTO syntax to automatically generate field name heading labels for the fields on the SUMMARY *CSVAUTO line, the aggregation operator will be appended to the field name heading label after an '_' (underscore) rather than a '/' (slash), since many applications accepting CSV input discard a '/' in a field name (or label).

So if you e.g. have:

```
TOTAL EOF AMT AMT/AVG
SUMMARY *CSVAUTO AMT/V AMT/AVG
```

you may expect to se a heading like:

```
AMT_V,AMT_AVG
```

If you reference a created field (from a CREATE statement after the TOTAL statement) in the SUMMARY *CSVAUTO statement, no aggregation opertaor is appended to the field name heading label.

Observe that if you want to write multi-purpose reports, i.e. reports that can be used to produce "normal" and CSV (and possibly XML) output you will have to precede each CSV output line with the *!CSV! tag. These reports have to be run with the -CSV command line switch. Also, since all "normal" output statements are discarded when run with the -CSV switch you will also have to include a *!CSV! TOTAL statement in front of the *!CSV! SUMMARY *CSV statement, since any un-tagged TOTAL statement will be discarded.

16. The *CSV[AUTO] * syntax will output all fields known to the TOTAL paragraph, including all key/sort fields down to this break level (.i.e. if you break on key 2, key 1 and key 2 will be included, while no key fields are included for TOTAL EOF).

7.11 PREVIEW Section

The PREVIEW section is similar to a SUMMARY section in that its role is in relation to a control break, and the PREVIEW section is similar to the DETAIL section in that it is concerned with data from an individual record. In effect, the PREVIEW section operates as a DETAIL section that is applied only to the **first** record in a subtotal run. This is in contrast to SUMMARY sections which always follow the DETAIL lines that they summarized. This "previewed" record is printed before the actual DETAIL of the records that make up the particular subtotal run. The PREVIEW section may not be used if automatic formatting is active.

For example, if we had the following DEF for a (simplified) payroll file.

```
*   PAYROLL.DEF
*
MAS 3000
DEPT   X999   KEY1  "department number"
EMPL   X99999 KEY2  "employee number"
NAME   A30          "employee name"
ANSALY D            "annual salary"
```

We wish to produce a report with a layout that shows the department number before the detail of each department. In other words, we wish to preview the department number.

```
          DEPARTMENT 010

00110     Jones, Bob          11,000
00120     Smith, Frank        12,000
00125     Wright, Bill         9,500
                              --------
                              32,500

          DEPARTMENT 030

00135     Green, John         14,500
00165     Thomas, Peter       10,500
                              --------
                              25,000

...
```

The report instruction file which generates the above report would be as follows.

```
REPORT SALARIES
FILE PAYROLL.MAS
SINGLE
DETAIL
     EM---     NAME------------ --ANSALY
END
TOTAL DEPT ANSALY
PREVIEW
BL
               DEPARTMENT D--
BL
END
SUMMARY
                                 --------
                                 --ANSALY
END
```

7.11.1 TOTAL EOF PREVIEW

PREVIEW sections are associated with TOTAL statements. A PREVIEW section following a TOTAL EOF statement will print only once; **after the HEADING** on the first page and **prior to the first DETAIL** record. This is useful in three different ways.

1. A TOTAL EOF PREVIEW may be used in place of a HEADING resulting in a "heading" on the first page only. This could be useful for preparing a memorandum or letter where the PREVIEW contained the "To:", "From:", etc. and the opening paragraph; the DETAIL listed line items; the TOTAL EOF SUMMARY contains totals, and the closing remarks.
2. A TOTAL EOF PREVIEW may be used in conjunction with a HEADING to supplement the "heading" on the first page. For example, a TOTAL EOF PREVIEW could contain all of the reasons why a record would be selected for printing on a report. This could be useful for printing error messages associated with error codes. The recipient of the report would not have to refer to supplementary documentation to understand the report and the additional information is only printed on the first page. Normal page headings may be concise to conserve space.
3. A TOTAL EOF PREVIEW may also be used with or without a HEADING to create a "cover sheet" for a report. This may be descriptive information as described in (2) above, but also might contain distribution information. For example:

```
Copy  Recipient    Location          Distribution Method

 1    John Doe     Accounts Payable        Deliver

  2    Jane Smith     Purchasing              Mail
```

To create a "cover sheet" effect, include enough blank lines at the end of the PREVIEW to cause a page eject before printing the first detail record. The first page of the report would then be a "cover sheet" and the data would begin on page 2.

7.12 SORT Statement

The SORT statement instructs REPORT to reorder the records from the input file according to specified sort fields. The new sort order applies only to the report presentation and does not change the input file. SORT fields can come from LINK, TABLE, and CREATE statements and from the input file.

```
SORT field1 field2 ...
```

The records are sorted in ascending order based on these fields, in major to minor sequence. That is "field1" is analogous to KEY1 in a DEF, "field2" is analogous to KEY2 in a DEF etc. To sort based on descending order of a field, append "/R" or "/DESC" to the field name.

```
SORT SALARY/R LASTNAME
```

This SORT statement reorders the records for use in the report first by SALARY/D2 from highest to lowest and then by LASTNAME/A20 in alphabetical order.

7.12.1 Relationship of SORT Statement to Other Statements

The SORT statement precedes the HEADING section. The SORT field names must be encountered by REPORT in the REP instruction file before the SORT statement is encountered. That is, fields defined in CREATE (see Section 7.13.1 "CREATE Statement"), TABLE (see Section 7.13.5 "TABLE Statement"), and LINK (see Section 7.13.4 "LINK Statement") statements, must be defined before the SORT statement in which they are used.

If a KEY statement (see Section 7.13.3 "KEY Statement") is present, it must be placed **before** the SORT statement. If a SELECT statement precedes SORT, only the selected records are reordered. Statements after the SORT statement operate on the records in the new sorted order.

CREATE statements placed before the SORT statement are evaluated both prior to the SORT pass, and again following the SORT pass. **IMPORTANT: Placement of certain types of CREATE statements before a SORT statement can produce errors in output.** CREATEd fields that accumulate values (for example, counter fields), placed before a SORT statement would accumulate each value twice, and therefore CREATE statements of this type should **always** be placed after the SORT statement.

LINK MULTIPLE (see Section 7.13.4.4 "LINK One To Many (MULTIPLE)"), and "NX$" fields (see Section 7.16 "Internal Field Names") cannot be used with the SORT statement.

The TOTAL statement specifies control breaks for subtotaling based on the new sort field order. All of the TOTAL options may be used.

For example, an input file keyed on employee number (EMP#) is reordered by department (DEPT) and last name (LASTNAME) as follows:

```
SORT DEPT LASTNAME
```

The following TOTAL statement will take a control break for each department counting the number of employees, the total salaries and the average salary per department.

```
TOTAL DEPT EMP#/E SALARY SALARY/AVG
```

7.12.2 Comparison of SORT Statement and SORT Command

In Section 7.13.4.1 "LINK Example" a real estate file keyed on customer account number is sorted into an index file ordered by the property address (LOCNAME LOCNO). The index file is then used as the input file for a street address report and fields are linked in from the original account file.

The SORT statement simplifies this operation by eliminating the need for an index file and by eliminating the LINK statement.

```
REPORT LOCATION
FILE RE.MAS
*
SORT LOCNAME LOCNO
* ---------------
HEADING
CE   TOWN OF ENFIELD
CE   PROPERTY OWNERS BY PROPERTY LOCATION
BL
              LOCATION           OWNER
4/40
     OWNER ADDRESS     VALUE
END
DETAIL
     -LOCNO LOCNAME------- NAME--------------
1/40
     OWNADR--------  ----GROSS
END
```

The real estate records are sorted by LOCNAME and LOCNO, street name and number, and presented in this new order in the DETAIL layout.

This would be an efficient approach if we ran this report only occasionally. However, if the report was run frequently, and especially if it was run with the KEY statement (see Section 7.13.3 "KEY Statement") to pick out particular property addresses, then we would only want to sort the file once (see Section 4.5 "SORT Example Creating an Index File"), saving the sorted property addresses in the index file. Note, the KEY statement cannot be used **after** the SORT statement.

The maximum number of records that REPORT with SORT can handle is 100,000,000 (one hundred million) by default. However, this limit can be set at any desired value on a system wide basis. You might want to set a limit to prevent taxing system resources by sorting huge numbers of records. Assign the maximum number of records for REPORT SORT to the system logical name ADM$REPSRT, as in the following example:

```
> admlcr adm_repsrt 60000
```

Keep in mind that sorting a very large number of records inside REPORT via the SORT startment may be less efficient than pre-sorting the file prior to running the REPORT using the sorted output.

7.12.3 Conditional SORT Statement

SORT statements in REPORT can be executed conditionally at run time, controlled by the reply to a parameterized prompt. Use the following syntax:

```
SORT <parameter> = value sortfield1...
```

If the response to the parameterized prompt matches the value specified in the SORT statement, the SORT is performed. If not, the SORT statement is ignored. For example:

```
SORT <List by last name (L) or number (N)> = L LNAME
 SORT <List by last name (L) or number (N)> = N EMPNO
```

ordered either by last name or employee number, controlled by the response to a single run-time prompt.[17]

7.13 Processing Statements

This section describes the use of processing statements in a report. The statements include CREATE, SELECT, KEY, LINK, TABLE, RECODE, and EXECUTE. The placement of a processing statement in the REP instruction file determines the order of the execution. The statements may be in any position in the REP instruction file as indicated in the outline (Section 7.1 "Outline of a Report Instruction File (REP)") and should be put in the order the user desires the statements to be executed. The usual position for processing statements is between the HEADING section and the DETAIL section which means the statements are processed after the record is read and before the detail is printed. The statements might be placed before the HEADING section if they influence the data being printed in the heading. The CREATE statement may be included in a TOTAL paragraph, but then may only operate on fields that may be used in the SUMMARY paragraph. SELECT and KEY statements may not appear after a TOTAL.

The order of the processing statements can have a significant impact on performance and even the accuracy of the report. For example, if a report contains a LINK statement and a SELECT statement, placing the SELECT statement first would improve throughput because the link operates on the selected records only. However, if the SELECT statement is based on fields obtained via the link, then the LINK statement must precede the SELECT statement to obtain accurate results.

17. Note that because the substitutable parameter is identical in both SORT statements only a single prompt will be generated. The reply will be substituted into both locations (see Section 7.14 "Parameterization").

7.13.1 CREATE Statement

The CREATE statement is used to create a virtual field that is then usable as if it were an actual field. The field name of the newly created field must be unique, i.e., not included in the DEF of the file referenced in the FILE statement and not included in a previous processing statement. The syntax of the CREATE statement is:

```
CREATE new-fieldname/type [expression]
```

For example:

```
CREATE NEWFLD/A10

CREATE XFUND/X9999 0100

CREATE MSG/A7 'Overdue'

CREATE TAXAMT/D2 6.75

CREATE NET/D GROSS - EXEMPT

CREATE DUP/D2 IF JULY GT 50.00 THEN JULY / 2 ELSE JULY END

CREATE MSG/A8 IF PAID LT DUE THEN 'OWES' ELSE :
  IF PAID GT DUE THEN 'CREDIT' ELSE ' ' END
```

From these examples we can see that CREATE may be used to create constants, perform conditional computations or create an alphanumeric string for display. Note that the CREATE may extend beyond one line using the "colon" continuation convention. Also note in the above example that nested IF conditionals in a CREATE have only **one** terminal END.

A reference to a Data Dictionary element may be substituted for the field type specification as described in Section 1.4.5, e.g.

```
CREATE XITEM/@ITEM IF PO# NE ' ' THEN ITEM ELSE '99' END
```

The CREATE statement is typically used before the DETAIL statement to create values for individual record printout or after a TOTAL and before its SUMMARY. The CREATE may precede a SELECT (described below) to set up a virtual field to be used by the selection criterion. Alternatively, the CREATE may follow the SELECT, in which case the value is only "computed" after a particular record is actually selected.

A CREATE statement before the DETAIL section may be used to hold values from record to record for use in creating other fields. For example, a repeating description could be printed only when the value changed using the following example:

```
...
CREATE XDESC/A20 IF FUND NE XFUND THEN FDESC ELSE ' ' END
CREATE XFUND/X999 FUND
DETAIL
    XDESC-------------- ...
...
```

7.13.1.1 CREATE Statements after TOTAL Statements

CREATE statements used after a TOTAL statement are subject to the same restrictions that apply when designating fields for a SUMMARY paragraph, i.e. they can refer only to fields introduced in or after the last TOTAL statement. These include:

1. any field in the line of sort up to and including the control field in the last TOTAL statement. For example, if the control field is the KEY2 field, then the KEY1 and KEY2 fields may be used. In order to use the KEY3 field it must be placed on the TOTAL statement with a subtotaling operation.
2. any field used in a subtotal operation in the last TOTAL statement. Subtotaled fields in CREATE statements should **always** be referenced by appending the subtotaling operation suffix from the TOTAL statement to the field name (e.g. DESCRIPTION/FI, COST/V). Use the suffix in the CREATE statement even if it is not explicitly used in the TOTAL statement.[18] Without the subtotaling suffix REPORT cannot reliably identify the field in its internal field name table. See the example below.
3. any field introduced by a previous CREATE statement that occurs after the last TOTAL statement.

In the example below note that the subtotaling suffix is necessary to reference TIME/V in the CREATE expressions and in the summary paragraph even though it is not explicitly present in the TOTAL statement.

```
TOTAL TOPIC TIME/AVG TIME
CREATE TFLAG/A12 IF TIME/V GT 0.99 THEN :
            'EXCESS TOTAL' ELSE ' ' END
CREATE AFLAG/A12 IF TIME/AVG GT 0.40 THEN :
            'EXCESS AVG' ELSE ' ' END
SUMMARY
      ---------Total------Avg--------------------------
      TOP--- -TIME/V -TIME/AVG AFLAG------- TFLAG-------
      ------------------------------------------------
END
```

7.13.2 SELECT Statement

SELECT is used to select records for processing, i.e. printing or subtotaling. The SELECT is usually placed between the HEADING section and the DETAIL section. However, as mentioned before it may precede the HEADING section if it would influence the data printed in the heading. SELECT may not appear after a TOTAL.[19] The syntax of the SELECT statement is as follows:

```
SELECT logical-expression
```

18. If no suffix is present the default subtotaling operation for numeric fields is "/V". For all other fields it is "/FI". See Section 7.7 "TOTAL Statement". When you use RECODE in TOTAL statements make sure that all RECODE fields in the TOTAL statement precede the non-RECODE (subtotal) fields.
19. To prevent a summary from printing use the conditional SUPPRESS statement, described in Section 7.18.1 "SUPPRESS Statement".

Field names in the logical expression must either be defined in the file or have been previously set up by CREATE, LINK, or TABLE. For example:

```
SELECT CK BET 1000 AND 2000

SELECT EMPLDA GT 01-JAN-70

SELECT OBJ EQ 00 AND :
  APPR GT 10000
```

The "colon" convention for continuation lines is provided in SELECT. For more information on expressions see Chapter 8: “Expressions”.

If more than one SELECT statement appears in a REPORT they are logically "AND-ed", i.e. only those records that meet **all** the select criteria are processed.

7.13.2.1 ORSELECT Statement

Use ORSELECT statements to logically "OR" selection criteria in situations where the criteria must be given in multiple statements. REPORT converts any number of **consecutive** ORSELECT statements into **a single SELECT statement,** where the ORSELECT expressions are joined with OR.

The ORSELECT statement is designed especially to make parameterized reports more flexible and easier to use; but it can be used without parameterization.

The following example shows a typical application of ORSELECT. A repeating parameter[20] is used to prompt for an open-ended series of customer numbers to be selected. The object is to select records that match any of the customers numbers entered to one of the prompts.

```
ORSELECT CUST# EQ <<Enter Customer Number>>~
```

There is no limit on the number of consecutive ORSELECT statements.

7.13.2.2 No Record Selected

When a KEY or SELECT statement results in no record being selected for output, AdmReport may not output anything. If output is to a file, no file would be created.

Reports with at least one summary will produce output (the summaries and the heading, if any) even when no records are selected. This output can be suppressed by putting "o" (lowercase) in the string assigned to the logical name OPTION (see Appendix A: “Options”).

FORCE_HEADING (see Section 7.17.13 “FORCE_HEADING Statement”) will output the heading even when "o" is in OPTION.

When no records are selected for output (i.e. by combinations of SELECT, ORSELECT, or KEY) REPORT creates a logical name, ADM$NONE_SELECTED, set with the value "Y". This logical name is deleted when AdmReport starts, and will only be exist if no records are selected when the report finishes.

Note that if the main report file is empty, AdmReport will not generate any output and will never create a file (FORCE_HEADING has no effect when the main file is empty), and will not set the logical name ADM$NONE_SELECTED.

20. See Section 7.14.1 “Repetitive Parameterization”

7.13.3 KEY Statement

The KEY statement is used to select records where the selection is by values in key field(s) and can therefore use the direct-access mechanisms.[21] There may only be one KEY statement in the report. If a SORT statement (see Section 7.12 "SORT Statement") is included in the report, the KEY statement must precede the SORT statement. KEY must precede any TOTAL statement. The syntax of the KEY statement is as follows. (Although two lines are necessary to show the syntax of the statement, the actual statement must be on one line.)

```
KEY key-field1 key-field2 etc. VALUE value1 value2 etc.
   AND/TO value1 value2 etc.
```

21. Direct access is a method for locating a record in a file by searching the internal index of the file for specific key values. This is a more efficient method for searching for a record than using the SELECT statement which must read all of the data records in the file. See Section 2.4.3 "Sort and Access Control" and Appendix E: "File Concepts".

The word KEY is followed by the names of key fields. These must be key fields in the report input file, and they must be in major to minor order. After the key field name(s), the value(s) are specified, using either constants or special "KEY$" logical names.[22] Each KEY$ logical name can hold one key value. KEY$ and constant values can be mixed in the same KEY statement. If there are more sets of values, they appear separated from each other by "AND" or "TO". "AND" means find each of the records whose key field(s) have value(s). "TO" means find all records whose key field(s) have value(s) that lie between the value(s) to the left and right of the "TO".

For example:

```
KEY EMPL VALUE 00135

KEY EMPL VALUE KEY$EMPL

KEY EMPL VALUE 00135 AND 00172 AND 00245

KEY EMPL VALUE 00135 TO 00245 AND KEY$EMPL

KEY FUND DEPT VALUE 01 075

KEY FUND DEPT VALUE 00 060 TO 00 350

KEY FUND DEPT EMPL VALUE KEY$F KEY$D1 0 TO KEY$F KEY$D2 99999
```

A KEY selection criteria may encompass only part of the actual key fields in the file, although KEY must always begin with the major key which can then be followed by minor keys. An example showing a KEY select on partial keys is included in Section 7.13.4.2 “Interaction of LINK and KEY Statements”.

If there is more than one record with the specified key values, KEY works as follows:

```
KEY EMPL VALUE 00135
```

Selects only the first record for EMPL 00135.[23]

```
KEY EMPL VALUE 00135 AND 00208
```

Selects only the first record for EMPL 00135 and EMPL 00208

```
KEY EMPL VALUE 00135 TO 00135
```

Selects all EMPL 00135 records.

```
KEY EMPL VALUE 00135 TO 00135 AND 00208 TO 00208
```

Selects all EMPL 00135 and EMPL 00208 records

22. Since the KEY values determine which records REPORT will process, KEY$ logical names are translated before REPORT begins processing records. Therefore, KEY$ logical names must be assigned before REPORT is run. If a report is only being compiled (see Section 7.21 “Pre-Compiled Reports”), the KEY$ logical names do not need to be assigned.
23. Note, there may be several records with 00135 in the EMPL key field.

7.13.4 LINK Statement

The LINK statement is used to obtain field values in files other than the file named in the FILE statement in order to include these linked fields in the report. The file named in the FILE statement could be an index file, where data fields from the master file are linked into the report via the LINK statement. The syntax for the LINK statement[24] is as follows. (Although two lines are necessary to show the syntax of the statement, the actual statement must be on one line.)

```
LINK link-field-name(s) FROM link-file-name
   KEY IS/KEYS ARE key-field-name(s)
```

For example:

```
LINK DESC FROM FUND.TAB KEY IS FUND

LINK FUND ACCT OBJ FROM TCODE.TAB KEYS ARE T1 T2
```

The word LINK is followed by the names of the fields to be linked from another file. Field names that appear in this part of a LINK statement are then usable in the remainder of the report as if they were present in the actual record. In other words, LINK is another way to "CREATE" virtual fields for a report. The values for the virtual fields are supplied via the link procedure.

Following the link-field-names is the word FROM and the name of the link file. The field names being linked must all exist in this link file. Finally, after the KEY IS/KEYS ARE phrase are the names of fields already defined in the report that are to be used to form an identification which is treated as the key into the link file.

There is an option to assign another name (rep-field-name) for use in the report to fields created via the LINK statement. This is necessary if the link-field-name already exists in the report. The syntax for renaming link fields is as follows:

```
LINK rep-field-name(s) IS/ARE link-field-name(s)
FROM link-file-name KEY IS/KEYS ARE key-field-name(s)
```

For example:

```
LINK NEWDESC IS DESC FROM FUND.TAB KEY IS FUND
```

The field DESC from the data file FUND.TAB is renamed to NEWDESC for use in the report.

Finally there is an option to access the link file via the record position of the record in the link file. This is done by including the "relative-pointer-name" in parentheses after the key-field-name(s). This option is completely described in Section 7.13.4.1 “LINK Example”. The complete syntax of the LINK statement is as follows:

```
LINK [ rep-field-name(s) IS/ARE ] link-field-name(s)
    FROM link-file-name KEY IS/KEYS ARE key-field-name(s)
    [ (relative-pointer-name) ]
```

The LINK statement does not support a continuation line. Use the LINK paragraph syntax when linking a large number of fields (see Section 7.13.6 “LINK and TABLE Paragraphs: Alternative Syntax”).

24. There is an alternative LINK "paragraph" syntax that provides the same functionality as the LINK statement. You may find the LINK paragraph syntax easier to use and maintain, especially when LINKing a large number of fields (see Section 7.13.6 “LINK and TABLE Paragraphs: Alternative Syntax”).

There may be several LINK statements with the same or different file and/or key names. That is, the input file to the report may contain more than one pointer to another file (e.g. a pointer to a husband's record and his wife's record), or the report record may contain pointers to several different files. In either case the report input records are read by REPORT (as per the FILE statement) and the related fields in the target files are pulled into the "virtual" report record via the LINK statement, whereupon they can be manipulated as part of the same report record.

If the LINK is unsuccessful, the linked fields are set to null values. As always when accessing files by key value, the LINK file must be in sort order for the link to succeed.

7.13.4.1 LINK Example

For example, in a real estate assessment application we wish to print the owner's name and address, and the assessed value of a property from a real estate record. We wish the printout to be ordered by the location (address) of the property.

First, we must build an index to the file via the following file definition, into which we SORT the real estate file.

```
*     STREET.DEF
*
IDX 15000
LOCNAME A24 KEY1        "name of street"
LOCNO I KEY2            "number of street"
ACCT# XA99999           "real estate acct no."
```

The following report instruction file would produce the desired result. Note the report reads the index file as its input file, and uses the index file to access records in the master file via the LINK statement.[25]

```
REPORT LOCATION
FILE STREET.IDX
HEADING
CE TOWN OF ENFIELD
CE PROPERTY OWNERS BY PROPERTY LOCATION
BL
           LOCATION           OWNER
4/40
     OWNER ADDRESS    VALUE
END
LINK NAME OWNADR GROSS FROM RE.MAS KEY IS ACCT#
DETAIL
     -LOCNO LOCNAME------- NAME------------
1/40
     OWNADR-------- --GROSS
END
```

A sample of the report printout might be as follows:

```
    LOCATION           OWNER         OWNER ADDRESS    VALUE

  1 ABBE ROAD      SMITH, PETER     17 ELM STREET   20,700
  3 ABBE ROAD      JONES, HAROLD    3 ABBE ROAD     24,300
...
114 AMES STREET    WALLACE, FRANK   114 AMES ST     31,600
116 AMES STREET    BILLINGS, STEVE  2 CARTER AVE    35,300
...
 82 WARREN DRIVE   FRANKLIN, AL     48 ENFIELD ST   41,000
 88 WARREN DRIVE   JAMES, HAROLD    7 BROADWAY      32,600
```

25. Section 7.12.2 "Comparison of SORT Statement and SORT Command" presents a similar example which uses a SORT statement rather than an index file and a LINK statement.

7.13.4.2 Interaction of LINK and KEY Statements

To illustrate the use of the KEY statement in relation to LINK and index files we could insert the following line just before the LINK statement in the above "LOCATION" report.

```
KEY LOCNAME VALUE <STREET NAME> TO <STREET NAME>
```

By using parameterization, described completely in Section 7.14 "Parameterization", we have created a report that can list the owners and values for any given street, without reading either the entire index file (STREET.IDX) or master file (RE.MAS).

Note the use of the KEY statement on only part of the STREET.IDX key, namely the street name and not the street number.

7.13.4.3 LINK with the NULL Keyword

Typically, REPORT bypasses the LINK statement when the key(s) for a particular record is "null". That is, the LINK is not even tried and the link fields are set to null values.

The "NULL" instructs REPORT to try the LINK even if the key value(s) is null. There may be a valid record in the link file with null keys. For example:

```
LINK NULL ARRESTDA FBINO DOB FROM HIST.MAS KEY IS CRIMID
```

7.13.4.4 LINK One To Many (MULTIPLE)

LINK can be used to link a single report record to multiple target records. This is typically done by supplying a part of the target key in the LINK statement. For example, if we have an incident record keyed by case number, and a file of people involved in an incident keyed by both case number and a sequence number:

```
*     CASE.DEF
      MAS 1000
      CASE XA99999 KEY1     "Case number"
      DATE DA               "Date when the incident occurred"
      TIME A8               "Time when the incident occurred"
      LOCATION A20          "Location where the incident occurred"
      OFFICER A20           "Officer reporting the incident"

      *     PERS.DEF
      MAS 5000
      CASE XA99999 KEY1     "Case number"
      SEQ I KEY2            "Sequence number for this record"
      LNAM A20              "Last name of the person involved"
      FNAM A10              "First name of the person involved"
      ADDR A20              "Address of the person involved"
      CITYST A20            "City and state of the person involved"
      ROLE A1               "Role of the person in the incident"
      ID XA99999            "ID of the person if on file"
```

We wish the report to read cases, print something about them, and then show something about the people involved. We use the word MULTIPLE in the LINK statement for PERS.MAS to instruct REPORT that CASE is only a partial key in PERS.MAS, and that there may be several records in PERS.MAS for each CASE in CASE.MAS. (The TABLE statement is described in Section 7.13.5 "TABLE Statement" below.)

```
REPORT CASE
FILE CASE.MAS
HEADING
CE   CASES AND PERSONS INVOLVED
BL
END
LINK MULTIPLE SEQ LNAM FNAM ROLE ID FROM PERS.MAS KEY IS CASE
TABLE ROLENM IS DESC FROM ROLE.TAB KEY IS ROLE
TABLE OFFNAME FROM OFFICER.TAB KEY IS OFFICER
DETAIL
     -SEQ LNAM---------- FNAM------ ROLENM----- ID----
END
TOTAL CASE DATE/FI TIME/FI LOCATION/FI OFFNAM/FI
PREVIEW
     ------------------------------------------------------
     CASE--- DATE-----  TIME------ LOC--------- OFFNAM------
BL
END
```

Sample printout might be as follows:

```
-------------------------------------------------------
R00753  03-DEC-75  15:33:00   300 MAIN ST  PTL JONES

   1 PETERSON        ROBERT     VICTIM
   2 JAMES           FRANK      WITNESS
   3 WILLIS          HARRY      PERPETRATOR  M00736

-------------------------------------------------------
T00736  03-DEC-75  15:42:00   425 ELM ST   PTL MINOR

   1 ANGELO          BURT       DRIVER
   2 GRIFFIN         PETER      PASSENGER
   3 MCCORMICK       KATHY      WITNESS
   4 FIELDS          FRANK      WITNESS
```

When doing a LINK MULTIPLE, REPORT **reprocesses** each record from the input file for **each** repeating link record in the LINK MULTIPLE target file. There may be only **one** LINK MULTIPLE per report. LINK MULTIPLE may not be used in a report containing a SORT statement (see Section 7.12 "SORT Statement").

7.13.4.5 LINK Without an Exact Match

The LINK statement will either find an exact match in the link file or, if no match is found, will return with null values for the link fields. Four alternative linkage operations are also available when an exact match may not be found but when an actual link is desired. These operations compare the link key values to the key values in the link file and link to the next higher or lower record in the link file when there is no exact match, or even if there is an exact match.

1. LINKGT - Link Greater Than: Links to the next higher record in the link file even if there is an exact match. If there is none higher, null values are returned for the link fields. This happens when the link key values are equal to or exceed the last record in the link file.
2. LINKGE - Link Greater Than or Equal to: Links to an exact match, or if one is not found, links to the next higher record in the link file. If there is none higher, null values are returned for the link fields. This happens when the link key values exceed the last record in the link file.

3. LINKLT - Link Less Than: Links to the next lower record in the link file even if there is an exact match. If there is none lower, null values are returned for the link fields. This happens when the link key values are lower than or equal to the first record of the link file.
4. LINKLE - Link Less Than or Equal to: Links to an exact match, or if one is not found, links to the next lower record in the link file. If there is none lower, null values are returned for the link fields. This happens when the link key values are lower than the first record of the link file.

The following schematic example illustrates these link operations. The REPORT main file is NULL.MAS, which contains 10 records keyed by integers from 1 to 10. The link file, ODD.MAS, contains 6 records and is keyed by odd integers ranging from 1 to 11. The REPORT links from NULL.MAS to ODD.MAS using the five link operations, linking in the appropriate SP (the number spelled out) descriptive fields.

```
* NULL.DEF  Data in NULL.MAS     * ODD.DEF   Data in ODD.MAS
* --------  ---------------     * -------   ---------------
MAS 100     I     NULL          MAS 100     N  SP
I I  KEY1   -     ----          N I KEY1    -  --
NULL A1     1                   SP  A10     1  ONE
            2                               3  THREE
            3                               5  FIVE
            4                               7  SEVEN
            5                               9  NINE
            6                               11 ELEVEN
            7
            8
            9
            10

* DEMO.REP
*
FILE NULL.MAS
SINGLE
HEADING
     DEMO OF LINK, LINKLT, LINKLE, LINKGT, LINKGE
BL
          LINK      LINKLT    LINKLE    LINKGT    LINKGE
          --------  --------  --------  --------  --------
END

LINK SP1 IS SP FROM ODD.MAS KEY IS I
LINKLT SP2 IS SP FROM ODD.MAS KEY IS I
LINKLE SP3 IS SP FROM ODD.MAS KEY IS I
LINKGT SP4 IS SP FROM ODD.MAS KEY IS I
LINKGE SP5 IS SP FROM ODD.MAS KEY IS I
DETAIL
     I-   SP1-----  SP2-----  SP3-----  SP4-----  SP5-----
END

REPORT RESULT
=============
DEMO OF LINK, LINKLT, LINKLE, LINKGT, LINKGE

     LINK      LINKLT    LINKLE    LINKGT    LINKGE
     --------  --------  --------  --------  --------
1    ONE                 ONE       THREE     ONE
2              ONE       ONE       THREE     THREE
3    THREE     ONE       THREE     FIVE      THREE
4              THREE     THREE     FIVE      FIVE
5    FIVE      THREE     FIVE      SEVEN     FIVE
6              FIVE      FIVE      SEVEN     SEVEN
7    SEVEN     FIVE      SEVEN     NINE      SEVEN
8              SEVEN     SEVEN     NINE      NINE
9    NINE      SEVEN     NINE      ELEVEN    NINE
10             NINE      NINE      ELEVEN    ELEVEN
```

The LINK statement only returns values when there is an exact key match between the main file and the link file. The LINKLT command only returns the SP value for the next lower keyed record in the file. Since there is no record in ODD.MAS with a key value less than 1, no SP value is returned for the key value of 1 in the main file. In this example, the LINKLE command always returns a value since it operates either using the exact key match (on the odd numbers in the main file), or by linking to the next lower key value (the even records in the main file). The same principles are applicable to the LINKGT and LINKGE operations.

7.13.5 TABLE Statement

The TABLE statement is used to create a field (or fields) by performing a look-up on a table, which is itself an ADMINS file. The LINK and TABLE statements are quite similar. However the internal programs behind each one are optimized for different purposes. LINK should be used when the record in the report and the record being looked up are more or less in one-to-one correspondence, whereas TABLE should be used when many report records can look up the same table record, i.e. when there is a genuine table involved.

Typical uses of TABLE are to get meaningful descriptions for code values, e.g. a vendor name and address for a vendor number, or to get table values for an identified item, e.g. a reorder point for a commodity. The syntax[26] for the TABLE statement is the same as the LINK statement and is as follows. (Although two lines are necessary to show the syntax of the statement, the actual statement must be on one line.)

```
TABLE [ rep-field-name(s) IS/ARE ] table-field-name(s) FROM
  table-file-name KEY IS/KEYS ARE key-field-name(s)
```

For example:

```
TABLE VENDOR CITYST ZIP FROM VENDOR.TAB KEY IS VEND

TABLE DEPTNAME IS DESCRIPTION FROM DEPT.TAB KEY IS DEPT

TABLE VALUE FROM USEDCAR.TAB KEYS ARE YEAR MAKE MODEL
```

There may be several TABLE statements in a report instruction file.

If a TABLE (or LINK) field is to be used in a CREATE statement or a SUMMARY layout following a TOTAL statement, the field must be made accessible via the TOTAL statement. A TABLE (or LINK) field is brought into a TOTAL paragraph by using the "/FI" or "/LA" function (see Section 7.7 "TOTAL Statement") in the TOTAL statement.

Unlike the LINK statement (see Section 7.13.4.5 "LINK Without an Exact Match"), the TABLE statement does not have options for accessing records when there is not an exact match between the key values and the table file. The complete key must be specified when using the TABLE statement. Table files (and link files) must always be in sort order.

26. As with LINK, there is an alternative TABLE "paragraph" syntax that provides the same functionality as the TABLE statement (see Section 7.13.6 "LINK and TABLE Paragraphs: Alternative Syntax").

7.13.6 LINK and TABLE Paragraphs: Alternative Syntax

There is an alternative syntax for LINK and TABLE statements, which is very similar to the LINK syntax in SCREEN (see Section 5.4.1 "LINK Paragraph"). A LINK may be specified in the following paragraph format:

```
LINK [NULL or MULTIPLE]  FILE_NAME
K KEY_FIELD
...
L LINK_FIELD [SECONDARY_NAME]
...
END
```

Similarly a TABLE may be specified in paragraph form, as follows:

```
TABLE FILE_NAME
K KEY_FIELD
...
L LINK_FIELD [SECONDARY_NAME]
...
END
```

The alternative "paragraph" syntax has advantages over the statement syntax in a variety of circumstances:

1. Paragraph syntax is much "cleaner" for specifying complicated links. Very long LINK or TABLE statements can be difficult to read, edit, and print.
2. Secondary names only need to be specified for the fields which need them. In the statement syntax, if one field needs a secondary name all fields must be given secondary names.
3. Unlike the statement syntax, there is no limit on the length of a LINK or TABLE paragraph, unless it is a LINK MULTIPLE.[27]

7.13.7 Automatic Field Renaming in LINK, TABLE Statements

Automatic LINK and TABLE field renaming makes it possible to write a very short LINK or TABLE statement, regardless of how many LINK (or TABLE) file fields the REPORT will use. This makes the .REP instruction file more concise, and makes it possible to change the REPORT without changing the LINK or TABLE statement, which is a convenient feature for developers. This feature can also make it easier to write heavily parameterized reports and other ad hoc reports where the specific fields used from a set of files may vary.

Syntax for automatic LINK or TABLE renaming is:

```
LINK =prefix FROM filename KEYS ARE key_fields

                    or

TABLE =prefix FROM filename KEYS ARE key_fields
```

This is identical to the usual LINK/TABLE syntax except that the list of fields and secondary names is replaced with an '=' followed by a prefix to be added to LINK/TABLE field names. Also, the keyword 'FROM' is not required (although it can be used) with this syntax.

27. The LINK MULTIPLE paragraph must translate into a LINK MULTIPLE statement of no more than 255 characters.

Any (or no) fields in the LINK/TABLE file can be referenced in the REPORT, or in the RMS as local fields, using the prefix. For example, if a LINK file has a field called 'FLD', and the prefix is 'A_', you can print FLD by referring to it in the DETAIL section as 'A_FLD'. You could also reference several other fields in this LINK file using the 'A_' prefix. The LINK statement does not change, regardless of how many LINK fields the REPORT actually uses (it may use none).

There can be several LINK or TABLE statements which use different renaming prefixes in the same report. Of course, the prefixes must be unique.

Automatically renamed LINK/TABLE fields can be used anywhere in REPORT that explicitly renamed LINK/TABLE fields can be used.

During the compilation phase of REPORT, all files named in LINK/TABLE statements are open simultaneously. However, during the execution of the report, the file named in an automatic LINK/TABLE statement is only opened if one or more fields in it are referenced in the REPORT. Only the fields which are referenced are actually linked in; so there is no performance penalty for using this feature.

The prefix string should be short, and when it is added to the LINK/TABLE file field names, the resulting names should be unique with respect to all the other fields involved in the REPORT. If not, REPORT uses the first field it finds with a given name. The prefix plus any field name in the LINK/TABLE file combined must be 18 characters or less in length. The prefix cannot begin with '-' or '$' and cannot contain a slash(/), a period (.), an apostrophe ('), or parentheses. A good prefix is a short string which ends with a delimiter character which is unlikely to appear in a field name: for example "A:".

When using the '=' syntax for automatic renaming, you cannot specify any explicit fields or secondary names in the LINK/TABLE statement.

LINKs or TABLEs using automatic renaming cannot appear below a TOTAL statement.

Automatic renaming is not available when using the alternative LINK/TABLE paragraph syntax in REPORT, described in Section 7.13.6 “LINK and TABLE Paragraphs: Alternative Syntax”.

7.13.8 RECODE Statement

The RECODE statement is used to create subsets for cross-tabulations based on a logical expression. "Recoded" fields can then be totaled[28] separately for each subset, and the subset totals individually designated for printing in the SUMMARY.

This is done by appending a ".n" to the print field designator in the SUMMARY paragraph, where "n" refers to the nth RECODE statement, i.e. ".1" refers to the subset created by the first RECODE statement, ".2" refers to the subset created by the second RECODE statement, etc. Up to 63 RECODE statements are supported.

28. RECODE cannot be used with a /AVG total field.

RECODE statements are best explained with an example. Consider the following file definition.

```
*   PAYROLL.DEF
*
MAS 1000
*
DEPT   X999   KEY1  "department number"
EMPL   X99999 KEY2  "employee number"
SEX    A1           "M=male F=female"
RACE   A1           "W=white O=other"
SAL    D            "annual salary"
```

Next, consider the following table:

```
    # of Employees, Minimum Salary, and Maximum Salary
              By DEPT, RACE, and SEX

                    WHITE                    OTHER
DEPT          MALE         FEMALE        MALE         FEMALE

100   #         17            4            2             1
     MIN     14,960         6,750       11,440         7,800
     MAX     18,350        12,575       17,225         7,800

200   #         45           10            6             2
     MIN     16,510         8,600       14,405         7,200
     MAX     19,970        16,550       16,775        10,500
```

The table shows that in department "100" there are 17 white males ranging in salary from $14,960 to $18,350.

The report instruction file that could produce this table from the file PAYROLL.MAS would look as follows.[29]

```
          MAX ----MAX.1   ----MAX.2   ----MAX.3   ----MAX.4
END     REPORT XTAB
FILE PAYROLL.MAS
SINGLE
HEADING
         # of Employees, Minimum Salary, and Maximum Salary
                   By DEPT, RACE, and SEX
BL
                       WHITE                  OTHER
     DEPT         MALE       FEMALE      MALE       FEMALE
BL
END
RECODE SEX EQ 'M' AND RACE EQ 'W'
RECODE SEX EQ 'F' AND RACE EQ 'W'
RECODE SEX EQ 'M' AND RACE EQ 'O'
RECODE SEX EQ 'F' AND RACE EQ 'O'
*
CREATE ECNT/I EMPL
CREATE MINSAL/D SAL
CREATE MAXSAL/D SAL
*
TOTAL DEPT RECODE ECNT/E RECODE MINSAL/MIN RECODE MAXSAL/MAX
SUMMARY
     DE--  #    -EC.1       -EC.2       -EC.3       -EC.4
          MIN ----MIN.1   ----MIN.2   ----MIN.3   ----MIN.4
```

29. The CREATEd fields ECNT, MAXSAL, and MINSAL are included in XTAB.REP to simplify identification of recoded fields in the SUMMARY paragraph. Designation of recoded fields in the summary paragraph and in CREATE statements after the TOTAL (see example below) can become confusing and sometimes ambiguous. Using unique CREATEd fields for each RECODE field in the TOTAL statement precludes any ambiguity and is recommended practice.

There is another method for specifying cross-tabulations which is particularly useful when there are a large number of cells in the table. Set up a prototype line and place an "R" in column one of that line. This instructs REPORT to generate the line for each RECODE statement. Using this method a SUMMARY paragraph for the table in the previous example could look as follows:

```
...
HEADING
1/22
        # OF       MINIMUM     MAXIMUM
        EMPL.      SALARY      SALARY
...
SUMMARY
     DEPT DE--
2/10
     WHITE MALE
     WHITE FEMALE
     OTHER MALE
     OTHER FEMALE
2/22
R      -ECN.1      -MIN/MI.1   -MAX/MA.1
6/1
BL
END
```

This instructs REPORT to produce four lines (corresponding to the four RECODE possibilities), at 2/22, 3/22, 4/22, and 5/22. Double spacing could have been instructed by specifying "R2" instead of "R", and then the lines of figures would appear at 2/22, 4/22, 6/22, and 8/22. The "Z" for zero suppress function can be specified with the "R" as in "ZR" and "ZR2".

The table produced by this alternative SUMMARY paragraph would look as follows. It contains the same information as in the other table, but presented in a different format.

```
                          # OF       MINIMUM     MAXIMUM
                          EMPL.      SALARY      SALARY

DEPT 100
         WHITE MALE         17        14,960      18,350
         WHITE FEMALE        4         6,750      12,575
         OTHER MALE          2        11,440      17,225
         OTHER FEMALE        1         7,800       7,800

DEPT 200
         WHITE MALE         45        16,510      19,970
         WHITE FEMALE       10         8,600      16,550
         OTHER MALE          6        14,405      16,775
         OTHER FEMALE        2         7,200      10,500

...
```

The R prototype line must contain only field designators for recoded fields, no literals or non-recode field designators should be included as they will not be repeated on the lines generated.

Care should be taken to reserve space for the output lines to be generated (one line per RECODE statement, two lines if double spacing). Note in the above example that literals were specified for lines 2 through 5 of the SUMMARY. These literals create the 4 lines for the output to be generated by the 4 RECODE statements, which cannot generate new output lines by themselves.

Created fields following a TOTAL statement may use recoded fields from that TOTAL.[30] This is done by subscripting the recoded field in order to access the recoded values. For example, to include the average salary for males and females (both other and white) to the report above, the following statements are added to XTAB.REP.

```
CREATE TOTSAL/D SAL
  ...
TOTAL DEPT RECODE ECNT/E RECODE TOTSAL RECODE MINSAL/MIN :
        RECODE MAXSAL/MAX
CREATE MAV/D   (TOTSAL(1) + TOTSAL(3)) / (ECNT(1) + (ECNT(3))
CREATE FAV/D (TOTSAL(2) + TOTSAL(4)) :
              / (ECNT(2) + ECNT(4))
SUMMARY
  ...
BL
     AVERAGE SALARY MALES      -------MAV
     AVERAGE SALARY FEMALES    -------FAV
BL
END
```

7.13.9 Quit Before the End of File: Q$Q

Normally REPORT processes the input file from beginning to end. However, there will be times when REPORT is only required to process the file up to a particular record and processing of subsequent records is not needed.

If a field Q$Q of type integer is CREATEd prior to the first TOTAL statement then Q$Q can be used to instruct REPORT to stop processing at a given record and act as if it reached the end of file, i.e. processing of all TOTAL statements occur, etc. The "CREATE Q$Q/I" statement should contain logic which sets Q$Q to "1" at the record where REPORT is to quit. The record where Q$Q = 1 is NOT processed and output by REPORT.

If Q$Q is CREATEd before a SORT statement, it ends the SORT pass as if the end of file has been reached. Thus Q$Q can be used to prevent REPORT from sorting the entire file when it is not necessary to do so.

30. When you use expressions after TOTAL statements make sure that all RECODE fields in the TOTAL statement precede the non-RECODE (subtotal) fields. Failure to do so could prevent REPORT from distinguishing non-RECODE subtotal fields from RECODE fields based on the same fieldname. See Section 7.13.1.1 "CREATE Statements after TOTAL Statements" for more information about using created fields after TOTAL statements.

7.13.10 Layout Statement: Variable Formatting

REPORT can print output in different formats, depending on Boolean expressions which are evaluated as the report runs. This ability to select alternative output formats based on the data provides additional flexibility in the formatting capabilities of REPORT.

Variable print formats are controlled using the LAYOUT statement, and a special '#' syntax in the DETAIL sections (or HEADING, PREVIEW, SUMMARY).

The LAYOUT syntax is:

```
LAYOUT   #n   Expression
```

Each LAYOUT statement is identified by a number (#n), and can control one or more pieces of the layout. The '#' notation in the layout section identifies the corresponding part(s) of the layout. For example:

```
LAYOUT #1 RECTYPE EQ 100
LAYOUT #2 RECTYPE EQ 200
HEADING
CE   Variable Formatting
#1
     Record Type 100 FLD1---- FLD2----
#2
     Record type 200 FLD6------------ FLD3----
#
CE   TODAY----
END
DETAIL
     First two lines of detail
     always contain this text.
#1
1/40
     Detail format for record type 100
#2
1/40
     Detail format for record type 200
     Any number of lines
END
```

The LAYOUT statements mean, for example, when RECTYPE EQ 100, then print using parts of the layout numbered "#1". That is, when the LAYOUT expression is true, use the correspondingly numbered parts of the layout; and when the expression is false, ignore those parts of the layout. The LAYOUT statement numbers are arbitrary except that they must be between 1 and 32767. LAYOUT can be placed anywhere in a report where a CREATE would be valid; but the LAYOUT statement must appear above (before) the first layout paragraph which it controls.[31] The syntax of the LAYOUT expression is the same as for SELECT. A colon (:) can be used to continue the expression on one or more additional lines.

To control part of the layout with the LAYOUT statement numbered '#n', place "#n" at the left margin on a line by itself above the part of the layout which "LAYOUT #n" will control. After the "#n" line, any number of output lines may be specified in the usual way. The end of the layout being controlled by "LAYOUT #n" is indicated either by another line beginning with "#", or by the END which ends the layout paragraph. In the example above, the part of the HEADING controlled by LAYOUT #1 ends at the "#2" line, which also begins the control of "LAYOUT #2". The end of "LAYOUT #2" control is denoted by a line with just '#', meaning that the next line(s) are not under the control of any LAYOUT statement.

A single LAYOUT statement can control any number of pieces of the layout, which can be in different layout paragraphs (there might be pieces of the layout numbered "#1" in both the HEADING and the DETAIL; they are both controlled by the LAYOUT #1 expression). There is no limit on the number of LAYOUT statements or on the number of pieces of the layout which a LAYOUT statement can control.

Parts of the layout which are not numbered with the '#' syntax are not controlled by a LAYOUT expression, and are always printed in the normal manner. In the example, the first and last lines in the HEADING paragraph are always printed.

For testing the layout, LAYOUT statements can be commented out, so all print layout sections they control will be included in the output.

---

31. It is not recommended that LAYOUT be used to produce output sections whose length can vary greatly, because this may cause unnecessary page ejects. REPORT always checks to see whether the next print section (DETAIL, PREVIEW, or SUMMARY) will fit on the current page; and if not, REPORT does a page eject before printing the next section. When REPORT makes this calculation, it does not "know" whether any lines in the section will be suppressed because they contain only blank fields; or because they contain blank floating fields; or because LAYOUT may suppress them. If the layout section specifies ten lines of layout, REPORT performs a page eject if it cannot fit ten lines on the current page, regardless of whether some of the lines are later suppressed. In this respect, suppression with LAYOUT behaves the same way as the suppression of lines of blank fields or floating fields. When different layout sections of varying length are used in a mutually exclusive way, this problem can be avoided by using precise placement, so that each section begins at the same line. When this technique cannot be used, DIRECT (see Section 7.17.14 “DIRECT Statement: Multiple Output Files”) can be used (DIRECT every section to the same spooler). Except when deciding whether to eject before a PREVIEW, DIRECT determines pagination according to the actual printed length of each section, not its maximum possible length.

7.14 Parameterization

The user issues the REPORT command, types the name of the report instruction file (REP), and may also type the name of the particular report within the REP instruction file. REPORT reads the source text of the REP instruction file, translating the statements in the REP instruction file into various internal tables and executable structures. Finally, REPORT reads the data file and begins to generate the actual report.

Parameterization is a feature that permits user-oriented editing of the report instruction file as it is being read and interpreted by REPORT. When the REP instruction file is prepared, strings in the report that are to be supplied by the user at run time are replaced by names, or "prompts", for these strings. These names or prompts are enclosed in "angle brackets". At run time, while the report instruction file is being read and interpreted by REPORT, these angle bracketed strings are given special treatment. Every time they are encountered, the text between the angle brackets is prompted[32]and the user is expected to enter text that replaces the angle brackets and their contents in the report text. Once text has been supplied for a particular parameter, i.e. a particular angle bracketed string, then that text will be substituted for the parameter each time it is encountered. For example:

```
*     DEPT.REP
*
REPORT SUMMARY
FILE BUDGET.MAS
SINGLE
HEADING
CE   SUBTOTAL <FIELD DESCRIPTION> BY DEPARTMENT
CE   PRINTED <PRINTING DATE>
BL
END
TOTAL DEPT <FIELD NAME>
SUMMARY
     DEPT-   DEPTNA-------------    -----<FIELD NAME>
END
TOTAL EOF <FIELD NAME>
SUMMARY
                                    ----------
                                    -----<FIELD NAME>
END
```

In the above report there are three parameters: FIELD DESCRIPTION, FIELD NAME and PRINTING DATE. FIELD NAME occurs four times in the report. The dialogue to run the report to summarize the appropriations by department would be as follows.

```
$report
REPORT FILE NAME: dept summary
FIELD DESCRIPTION: appropriations
PRINTING DATE: 24-sep-77
FIELD NAME: appr

          SUBTOTAL APPROPRIATIONS BY DEPARTMENT
                    PRINTED 24-SEP-77

020   MANAGER             42,075
030   TOWN ATTORNEY       18,150
...
420   POLICE             756,150
```

32.If the logical name ADM_DIALOGBOX is set to the value “P”, command files (AdmCom) and reports (AdmReport) will prompt in a dialog box, rather than in the command prompt window.

```
...
                                        ---------
                                        4,642,512
```

If a parameter is in single angle brackets (<>), and the user does not supply a run time string, i.e. the user presses return in response to the prompt, then REPORT will terminate. If however, the parameter is in double brackets, as in "SELECT <<type selection>>", and the user does not supply a run time string then REPORT will ignore the **entire instruction line** which contained the double bracketed string. Any other occurrences of this unresponded-to double bracketed parameter causes REPORT to ignore every line containing a copy of that parameter.

If the string enclosed in the angle brackets begins with the characters "L$" (e.g. <L$fieldname>), REPORT first attempts to translate this string as a logical name rather than prompting the user. Logical parameters are described in Section 7.14.2 "Logical Parameters".

Parameter values can optionally be changed (via RETRY) in an interactive dialogue as described in Section 7.14.4 "Rerunning Parameterized Reports, RETRY". Also parameter values can be substituted to be run at a later time, using the SAVE feature described in Section 7.14.3 "Saving Report Parameters for Later Use".

7.14.1 Repetitive Parameterization

If REPORT finds a ~ (tilde) to the right of the first right double angle bracket on the line (e.g. >>~), REPORT will continue to re-prompt that line until the user presses return to indicate that no more responses will be made.

```
SELECT <<TYPE 1 OR MORE EXPRESSIONS, C.R. WHEN DONE>>~
```

For this repetitively parameterized line, REPORT will continue to prompt for another SELECT expression until the user presses return.

```
FILE <DATA FILE NAME>
<<ENTER REPORT INSTRUCTIONS>>~
```

With this REP instruction file, a complete report can be composed with suitable responses to the repetitively parameterized prompt "<ENTER REPORT INSTRUCTIONS>".

While entering responses to report prompts, if the user decides to exit from the report (e.g. the previous response was incorrect) the backslash (\) character may be entered to any of the prompts. Pressing "\" exits from the report and closes all of the files opened by the report.

Repetitive parameterization can also be used with a series of logical names as described below.

7.14.2 Logical Parameters

If the parameter string contained in the angle brackets begins with the characters "L$", (e.g. <L$fieldname>), then REPORT first tries to translate the string as a logical name. If the logical name has been assigned in either the process, desktop or system logical name tables, the user is not prompted for the contents of the parameter. Instead the value of the logical name is substituted for the prompt. Parameters which begin with the characters "L$" and are assigned as logical names are called "logical

parameters". If a parameter beginning with "L$" is not assigned as a logical name, then the user is prompted for a value as in standard parameterization (see Section 7.14 "Parameterization").

Prompting for values when the logical name is not assigned can be avoided entirely by supplying a default value in the parameter string, as follows:

```
<L_MINIMUM=0>
```

Specify the default value for the logical name by appending ***"=value"*** to the logical name inside the angle brackets. In the example above if the logical name L_MINIMUM is not assigned, the value "0" will be substituted for the parameter.

When the logical names exist, the display of logical parameter prompts and their values can be suppressed by assigning the lowercase letter "c" to the logical name OPTION (see Appendix A: "Options").

If a logical name beginning with "L$" is used inside repetitive parameters (see Section 7.14.1 "Repetitive Parameterization"), then REPORT tries to translate a series of logical names by appending successive integers to the L$ logical name. For example, if <<L$LINK>>~ appears in a REP instruction file, REPORT tries to translate the logical names L$LINK1, L$LINK2, etc., until the logical name L$LINKn is not found. These values are substituted in the report and the user is not prompted. If the logical name L$LINK1 does not exist, then REPORT prompts the user for L$LINK1, L$LINK2, etc. until the user presses return to the prompt.

7.14.3 Saving Report Parameters for Later Use

REPORT has a facility to save the report instruction file after the responses have been entered for parameters so that the report can be run at a later time. The "saved" report can be run on-line or submitted to the batch queue.

If the keyword SAVE appears on the command line after the name of the REP instruction file and the (optional) report name, then, after the report is compiled successfully and parameter values, if any, have been substituted, REPORT creates a new REP instruction file with all parameters substituted, but does not actually run the report.[33] The general syntax for saving a report is:

```
$ REPORT REP_FILE_NAME [REPORT_NAME] SAVE [SAVE_NAME]
```

If SAVE is the last word on the REPORT command line, REPORT creates a new file named SAVE.REP which contains the report with substituted parameters. The user may supply a meaningful name for the "saved" report, by placing a first name for the "saved" report after the SAVE keyword. REPORT then writes SAVE_NAME.REP instead of SAVE.REP. If the save-name is the same as the report file name, REPORT creates SAVE.REP. It does not create a new version of the original REP instruction file, which could then be purged and inadvertently lost.

33. Any existing reports named "SAVE" will conflict with this syntax. For example, if the command REPORT BAL84 SAVE was intended to run a report named SAVE in BAL84.REP, it would instead now create a SAVE.REP which is a copy of the first report in BAL84.REP. Any reports named SAVE should be renamed. Note that this affects only the report name in the REPORT statement, not the name of the REP instruction file itself.

Let's look at a simple example of a parameterized report called PASTDUE.REP which prints a list of overdue invoices:

```
* Overdue invoices
REPORT PASTDUE
FILE INVOICE.MAS
SINGLE
SELECT INVDATE LE <INVOICE DATE>
SORT VENDOR INVDATE INVNO
HEADING
CE   Invoices Payable on or after <INVOICE DATE>
BL
END
DETAIL
     VENDOR----------------- INVDA---- INVNO----- --------NET
END
TOTAL VENDOR NET/V
SUMMARY
                                                          -----------
                                                          --------NET
BL
BL
END
```

The SAVE dialogue to create a version of the report named JUNE.REP, which will list invoices dated June 30, 1985 or earlier, is:

```
$ REPORT PASTDUE SAVE JUNE
INVOICE DATE: 30-JUN-85
JUNE.REP WRITTEN
```

The result, a new file named JUNE.REP, contains the following REPORT instructions:

```
* PASTDUE.REP saved on  5-SEP-85 at 12:46:36
*REPORT PASTDUE
FILE INVOICE.MAS
SINGLE
SELECT INVDA LE 30-JUN-85
SORT VENDOR INVDA INVNO
HEADING
CE   Invoices Payable on or after 30-JUN-85
BL
END
DETAIL
     VENDOR----------------- INVDA---- INVNO----- --------NET
END
TOTAL VENDOR NET/V
SUMMARY
                                                          -----------
                                                          --------NET
BL
BL
END
```

The resulting "saved" REP instruction file (e.g. JUNE.REP) contains the following modifications from the original REP instruction file.

1. A comment line inserted at the top of the instruction file which shows the original REP instruction file name and the date and time the report was saved.
2. The REPORT statement from the original report appears as a comment.[34]
3. All parameters have been substituted with the user supplied responses.
4. Comment lines (i.e. lines beginning with an asterisk "*") from the original report have been omitted.

The "saved" report can be run by simply requesting REPORT JUNE.

The SAVE facility can be used together with the RETRY feature described next in Section 7.14.4 “Rerunning Parameterized Reports, RETRY”.

7.14.4 Rerunning Parameterized Reports, RETRY

The RETRY facility in REPORT enables a user to immediately rerun a parameterized report and interactively change, or retain, one by one, the values of parameters which were previously supplied. RETRY may be used whether or not the report compiled successfully, so it can be used either to correct syntax errors in parameters, or to adjust the values of parameters in a report which ran successfully. Like the SAVE keyword (see Section 7.14.3 “Saving Report Parameters for Later Use”), the RETRY keyword must be given on the REPORT command line.

```
$ REPORT REP_FILE_NAME [REPORT_NAME] RETRY
```

RETRY is implemented by storing the values of parameters in the REPORT temporary file for reuse. A lowercase "r" must be present in the string assigned to the logical name OPTION (see Appendix A: “Options”) to enable the RETRY feature.[35] If "r" is in OPTION, REPORT does not delete the temporary file when it exits. Instead, RPxx.TMP will contain parameter values and other "compiled" information from the last time the report was run. (Note that only the last temporary file in a directory for a particular terminal number is saved.) Since parameter values are stored in the temporary file, and the temporary file is reused or recreated whenever REPORT is run, RETRY can only be used to rerun the last report which a user has run at a given terminal. If RETRY is specified on the command line but RPxx.TMP does not exist, REPORT prompts normally for parameters.

When RETRY is specified on the command line, REPORT finds RPxx.TMP. For each parameter, REPORT displays the value found in the temporary file and prompts the user. To keep the existing value, the user presses return. If anything other than return is entered, REPORT reprompts and the user may enter a new value for the parameter. When the RETRY dialogue ends, the report runs using the new parameter values.

34. The REPORT statement is optional for the first report in a REP instruction file. If no report name is supplied on the command line, REPORT defaults to the first report in the REP instruction file, whether or not there is a REPORT statement.
35. The REPORT temporary file has an automatically-generated name in the form RPxx.TMP. See C.1.1 “Differences in Print File and Temporary File Naming”.

If the response to an optional parameter (i.e. a parameter enclosed in double angle brackets, <<parameter>>) is a return, the RETRY dialogue displays CR as the current value. This may be retained by pressing return, or may be changed by entering a value for the parameter. On the other hand, if the original response to an optional parameter is a value, it can be changed to a return by entering the letters CR (meaning Carriage Return) as the new value in the RETRY dialogue.

If logical parameters are in use, (see Section 7.14.2 "Logical Parameters") and the logical names for those parameters are defined, then the RETRY dialogue displays the prompt and its value (the contents of the logical name) but **does not** give the user an opportunity to change the value.

Finally, SAVE (see Section 7.14.3 "Saving Report Parameters for Later Use") and RETRY may be used together, so that parameter values supplied in the RETRY dialogue can be SAVEd for later use, rather than being run immediately. The general syntax for using the RETRY and SAVE facilities together is:

```
$ REPORT REP_FILE_NAME [REPORT_NAME] RETRY SAVE [SAVE_NAME]
```

The following report dialogue using PASTDUE.REP from Section 7.14.3 "Saving Report Parameters for Later Use", shows how someone might create an overdue invoice report, after making a typing error on the first try. In this example, the SAVE facility is also used.

```
$ REPORT PASTDUE RETRY SAVE JULY
INVOICE DATE: 31-JUK-85
exp931  Error decoding constant
Line 5 INVDA LE 31-JUK-85
```

Again the report is run, this time correcting the typing error for the INVOICE DATE. Note that although the report did not compile successfully the parameter response was saved.

```
$ REPORT PASTDUE RETRY SAVE JULY
INVOICE DATE: 31-JUK-85  CR=OK? N
INVOICE DATE: 31-JUL-85
JULY.REP WRITTEN
```

Because the SAVE facility is used, the report is not run. Instead JULY.REP is written with the substituted parameter value for later use.

7.14.5 DEF Statement

The DEF statement displays the file size, number of fields, field names, field types and key designators on the user's terminal for one or more files. The information is taken from the data file directly, not from the DEF instruction file. The DEF statement may be placed anywhere in the REP instruction file outside of a section (e.g. the HEADING section). It is typically placed before parameterized FILE, CREATE, SELECT, LINK or TABLE statements:

```
DEF file1 file2 ...
```

For example, if a REP instruction file includes the following DEF statement,

```
DEF DEPTEMP.MAS
```

REPORT prints the following file definition information for DEPTEMP.MAS:

```
-------------------------------------------------
DEPTEMP.MAS has 3 fields, 100 records.
1. DEPT X99 KEY1  2. EMP# X9999 KEY2  3. SALARY D2
```

The DEF statement is intended to be used on-line when preparing parameterized reports and/or automatically formatted reports. See Section 7.14.1 "Repetitive Parameterization" for a description of repetitive parameterization and Section 7.5 "DETAIL Statement" for a description of automatically formatted reports. For example:

```
REPORT ADHOC
DEF <<Want to see some file definitions? Enter file name>>~
FILE <Enter file name>
```

7.15 Floating Fields

REPORT contains a feature called floating fields to handle the printing of a large number of fields many of which may be empty of data for any particular record being printed. The purpose is to allow a vertical column of fields in the report to be "collapsed" to significant (non-null) entries without gaps, and to allow associated literals to not be printed when the fields are null.

This is best illustrated with an example.

```
DETAIL
        ASSESSMENTS              EXEMPTIONS
1 8   (HOUSE    ---HOUSE)     (ELDERLY   ---ELD)
2 9   (LOT      -----LOT)     (BLIND     ---BLI)
3 10  (OUTBLDG ---OUTBL)     (VETERAN   ---VET)
4 11  (STORE    ---STORE)     (FARMER    ---FAR)
5     (FACTORY ---FACTO)
6     (ACRES    ---ACRES)
END
```

We are to print a record which contains assessment and exemption fields. A particular account usually has one or two assessment fields that are non-null, i.e. actual assessments. However, some records may contain more or even all of the assessments. Similarly, none, some or all of the exemptions may be active for a particular account.

Now let us examine the example. The numbers in the left margin of the DETAIL paragraph show the relation of the parenthesized "floating fields" that appear in the body of the DETAIL paragraph. That is the "1" applies to "(HOUSE ---HOUSE)", "8" applies to "(ELDERLY ---ELD)", "2" applies to "(LOT ---LOT)", "9" applies to "(BLIND ---BLI)", and so on.

Floating fields that are not one apart in their left margin number are fully independent of each other. That is, number the floating fields in a group consecutively and skip a number before starting the next group. In the above example, the ASSESSMENTS are represented by the set of numbers from 1 to 6 and the EXEMPTIONS are represented by the set of numbers from 8 to 11. No more than two sets of floating fields can be set up on **one line** of the report instruction file, but an output report line can be constructed from several lines in a report instruction file (e.g. by using the line/column notation). The numbers on the left margin can be between 1 and 63, and each number may be used **only once in the entire report,** that is, REPORT supports a maximum of 63 floating fields.

The logic of floating fields is as follows: If **at least one** data field enclosed in a pair of parentheses is non-null, the entire contents of the parentheses are printed. (The parentheses themselves are not printed.) If, however, **every** data field enclosed in a particular pair of parentheses is zero, the entire contents of the parentheses are **not** printed. When the contents of the parentheses are not printed, the fields in the next pair of parentheses "float" upward into the printout positions that would have been occupied by the zero fields.

For example, an account with house, lot, acreage, and a veteran and farmer exemption would print out as follows:

```
HOUSE    24,000          VETERAN   1,000
LOT       6,000          FARMER    5,000
ACRES    45,000
```

An account with a factory and no exemptions would look as follows:

```
FACTORY 130,000
```

7.16 Internal Field Names

REPORT provides several special internal field names that can be used in layout paragraphs as field designators. These are:

- PGNO for current page number.
- TODAY[36] for today's date.
- NOW[37] for current time (in military time HH:MM:SS).
- [TR] for total records printed thus far.
- [PR] for records printed this page.
- NX$fieldname, "lookahead", set to the value of field "fieldname" from the next record. **NX$fieldname cannot be used if a SORT statement is present in the REPORT.**
- NX$EOF,[38] "lookahead", an integer field that is set to -1 if the current record is the last record in the file. **NX$EOF cannot be used if a SORT statement is present in the REPORT.**

  [TR] and [PR] are particularly useful with the PAGE control break in TOTAL. The square brackets enclosing TR and PR are part of the syntax. When the internal field names are used to form print field designators, they must be spelled out fully.

7.17 Report Options

There are several options applicable to the entire report which are placed in the REP instruction file between the FILE statement and the HEADING statement. Any or all may be included as required.

36. TODAY may be used in REPORT processing statements and expressions without being declared. By default, TODAY is field type DA. If "d" (lowercase) is present in the string assigned to the logical name option, TODAY will be created as field type DT (see Appendix A: “Options”).
37. To be used in REPORT processing statements and expressions, (i.e. DETAIL statement, CREATE statement, SELECT statement, etc., NOW/A8 or NOW/TM must be declared, using either a CREATE statement without an expression or a local RMO field.
38. The NX$ fields that support the "lookahead" facility in REPORT should be declared by writing CREATE statements without expressions, or they can be RMO local fields.

7.17.1 SINGLE Statement

Report text is normally double spaced, that is REPORT places a blank line after the report output lines generated by each DETAIL record. The SINGLE statement instructs single spacing for DETAIL output, i.e. don't insert the extra blank lines between detail section output.

7.17.2 PAGE Statement

The PAGE statement can initialize the page counter (PGNO) to a value other than the (assumed) one of 1.

7.17.3 LENGTH Statement

The LENGTH statement specifies the number of lines to print on each page. The margin is the remainder when the length is subtracted from the number of lines per inch times the form length in inches. For example, to print 8 lines per inch on standard forms of length 11 inches with a standard 8 line margin the user would use LENGTH 80.

REPORT prepares output pages that are 60 lines in length by default. This length is intended for standard length forms (11 inch), standard 6 lines per inch printing, and a total top and bottom margin of 6 lines. All REPORT is actually doing is inserting a mechanical form feed between pages.

Any length paper or special forms may be used by resetting the page length with the LENGTH statement. LENGTH N sets the number of lines to be printed per page to N. After N lines are printed, a mechanical form feed is generated, and the report continues on the next page.

Printers normally have several settings for various form lengths and a switch which alters printing from 6 lines per inch to 8 lines per inch. The user is responsible that the LENGTH statement in the REPORT and the physical setting of the printer are consistent.

There may be situations when the physical length of the required form does not coincide with a length setting of the printer. If this is true then the mechanical eject feature of the printer should not be used. If N is negative, e.g. LENGTH -30, then REPORT assumes the output to be on continuous forms of non-standard length. Each form is assumed to be -N (30) lines long and REPORT s the printed output page to that form length using spacing and does not use the mechanical eject feature of the printer. Negative lengths are also useful when the printing device doesn't have mechanical form feed capability. Negative length may also be used when no margin is desired. For example, for a 3 1/2 inch form, printing 6 lines per inch and no margin, would require LENGTH -21.

If N is zero, LENGTH 0, then REPORT prints continuously without any page breaks.

7.17.4 WIDTH Statement

The default width for REPORT is 132 columns (80 columns for OUTPUT VT). Use the WIDTH statement to alter the width of the output line.

You may specify a WIDTH value up to 1023, both for queuing and direct printing. For example:

```
WIDTH 150
```

specifies a line width of 150 characters.

The INDENT value (see Section 7.17.6 "INDENT Statement") plus the WIDTH value may not exceed 1023.

7.17.5 IPAD Statement

The PAD statement acts like the WIDTH statement, but will also ensure that all report output lines will be padded with blanks to the width specified by the PAD statement. E.g.

PAD 376

will create an output file with exactly 376 characters in each line.

No *INDENT or WIDTH statement should be present if the PAD statement is used.*

7.17.6 INDENT Statement

Use the INDENT statement to set alternate indentations. e.g.

```
INDENT 10
```

means start printing the output lines in column 10.

7.17.7 OUTPUT Statement

The OUTPUT statement is used to instruct report where to direct the output of the report.

7.17.7.1 OUTPUT LP (Line Printer)

If the OUTPUT statement does not appear in the REP instruction file or the statement is "OUTPUT LP", the output will be directed (queued) to the device assigned to the logical name ADM$SPOOL0, or to the device number specified by the LP statement (see Section 7.17.8 "LP Statement"). If the designated ADM$SPOOLn logical name is not assigned, or if the queue designated does not exist, the output file is simply written to disk.

7.17.7.2 OUTPUT TI (User's Terminal)

If the statement is "OUTPUT TI" (or "OUTPUT KB"), the output will be directed back to the user's terminal and physical form feeds are issued between pages. By setting the characteristics of your terminal to perform form-feeds,[39] you can achieve perfect page to page spacing of reports.

Before REPORT begins sending output to the terminal it prompts

```
adjust paper
```

to allow the user to check alignment of the paper supply.

39. If the terminal is of a type that will recognize and execute a form-feed the setup command is "$ SET TERMINAL /FORM_FEED".

7.17.7.3 OUTPUT VT (Video Terminal)

One can also display the report at a video terminal and let the user read the report page by page. This is done by giving the REPORT command from a video terminal. The first page of the report is displayed, and then REPORT pauses until the user presses return. Then the next page is displayed and so on.

In order to invoke this option the user places the statement "OUTPUT VT" in the report instruction file.

REPORT automatically defaults to a WIDTH of 80 and a LENGTH of 24 when it reads OUTPUT VT. The user may alter these automatic settings by placing WIDTH or LENGTH statements after the OUTPUT statement.

7.17.7.4 OUTPUT LA (Terminal's Printer Port)

"OUTPUT LA" is similar to "OUTPUT TI" except that the output is directed to the "printer port" of DEC VT-compatible terminals[40].

7.17.7.5 OUTPUT SO (Direct Output to Standard Output)

"OUTPUT SO" is similar to "OUTPUT TI" except that no “adjust paper” prompt is displayed. Output begins to standard output as soon as REPORT can format it. Use this alternative to directly display REPORT’s output in a process where the standard output is not a terminal and/or is to be redirected or “piped”.

7.17.7.6 OUTPUT TT0 (Direct Output to Physical Device)

It is possible to send report output directly to any physical device.

If the statement "OUTPUT TT0" is used, the output is directed to the device assigned to the logical name ADM$PRT0 (see Section 21.4 “Output to Non-queued device, ADM$PRT0”).

7.17.7.7 OUTPUT DP (Output to the Default Printer)

OUTPUT DP routes report output directly to whatever is designated as the default printer when the report is run.

The setting of the logical name ADM$SPOOLn and/or the device number specified by the LP statement (see Section 7.17.8 “LP Statement”) are ignored when OUTPUT DP is used[41].

7.17.7.8 OUTPUT PD (Select Printer from DIalogue Box)

OUTPUT PD presents the user with a “printer dialogue box” when the output is ready for printing. The output is routed to the device selected using the printer dialogue box.

The same considerations noted in Section 7.17.7.7 “OUTPUT DP (Output to the Default Printer)” also apply when using OUTPUT PD.

---

40. Usually the "terminal" is a terminal emulator these days. Most Windows-based terminal emulators send "printer port" output to the Windows "default computer".

41.Having the end user choose the printing device (either via choosing the default printing device and having the output routed to the default printer, or by choosing a printer from a dialogue box) should not be a problem if you do not embed any special printer control "escape" sequences to obtain special printer effects (e.g. bolding a certain value or phrase), for example, by using an ADM_STYLE table and STYLE statements in your report.

If you do use the ADM_STYLE table (or another method) to embed printer control escape sequences in the report output, ***it is your responsibility*** to make sure those printer control sequences are appropriate for all printers that could be the destination device for the report.

This fact underscores the value of a homogeneous printer environment, for example, one where all printers support PCL5 printer control sequences, especially when OUTPUT DP (and OUTPUT PD) are made available to users, which allows them the flexibility to choose any device they have access to for their report output.

7.17.8 LP Statement

The LP statement controls the various options associated with reports directed to any printing device, usually a line printer. The LP statement should follow the OUTPUT LP statement.

When report runs and the output is to be to a line printer, report creates a file called "ADMINSxx.LIS" where "xx" is a unique, automatically-generated string.[42]

This file is then queued using the system queuing facilities. Spooling is discussed in detail in Chapter 21: "Printer Queues". The LP statement options are presented here.

The keywords of the LP statement are positional, i.e. if the second keyword is used then the first keyword must be included. The syntax of the LP statement is as follows:

```
LP [NCOPIES] [LOGICAL QUEUE#] [#OVERPRINTS] [NO/FORMS TYPE]
```

7.17.8.1 Multiple Copies

The first LP keyword specifies the number of copies to be printed.

```
LP [NCOPIES]

LP 2
```

The number of copies is an integer value placed immediately after the LP. If there is no LP statement, or if there is an LP statement with a 1 in the first position after the LP, one copy of the report will be printed. Multiple copies in this context does not refer to multi-part paper, but to the number of originals. Multiple copies of reports are only supported via queuing and are not applicable if the output is not queued by REPORT. Up to 99 copies can be requested.

7.17.8.2 Logical Queuing Device Number

The second keyword on the LP statement is the logical print queue number. The default queue number for a report is 0, i.e. the output is sent to the queue specified by the logical name ADM$SPOOL0.

The "LOGICAL QUEUE#" can be used to output the report to any queue. The output is sent to the queue assigned to ADM$SPOOLn where "n" is the logical queue#. This keyword has no effect if the output is not queued by REPORT.

```
LP [NCOPIES] [LOGICAL QUEUE#]

LP 1 7
```

This would send the output to the queue assigned to ADM$SPOOL7. Note that even though the "NCOPIES" keyword was not used, a "1" was included because the keywords are positional.

See Chapter 21: "Printer Queues" for more information on spooling.

42. Whenever an ADMINS command produces a ".LIS" file on Windows systems it assigns the automatically-generated name to the logical name ADM$LISTFILE in the "process" logical name table. This mechanism provides a way to determine the name of the file produced by the most recent step in a command procedure.

7.17.8.3 Overprinting

Overprinting is instructed via the third keyword on the LP statement. Overprinting consists of printing the same line on the printing device more than once. It is used to produce darker copies at slower printing speeds. Overprinting is not associated with queuing, but rather REPORT writes each output line multiple times.

```
LP [NCOPIES] [LOGICAL QUEUE#] [#OVERPRINTS]

LP 1 0 2
```

This would mean to print each line twice. Again, note the "1" for "NCOPIES" and the "0" for "LOGICAL QUEUE#" to position the "2" for "#OVERPRINTS".

7.17.8.4 Bypass Queuing; Specify Form Type, File Name

The fourth and fifth keyword positions of the LP statement are used for three purposes. REPORT can be instructed to bypass queuing its output file for printing, a forms type can be specified for a queued report, or the output file (normally ADMINSxx.LIS) for a queued report can be given another name.

The presence of the word "NO" as the fourth element of the LP statement instructs REPORT not to queue the report's output. The number of copies and the queuing device number keywords are only used as placeholders if the report is not queued by REPORT.

```
LP [NCOPIES] [LOGICAL QUEUE#] [#OVERPRINTS] [NO/FORMTYPE]
[FILENAME]

LP 1 0 0 NO
```

Alternatively the fourth element of the LP statement can be used to designate (by name or number) the form that should be used to print the document.

```
LP 1 2 0 6
```

The presence of a form name in the fourth position of the LP statement queues the output file with a specific form name.[43]

```
LP 1 2 0 PAYCHECKS
```

The LP statement above instructs REPORT to output 1 copy to ADM$SPOOL2 with forms type PAYCHECKS.

If a fifth keyword appears on the LP statement it is used to name the output file that will be queued for printing.[44]

```
LP 1 2 0 - ACCOUNTS.LIS
```

The LP statement above instructs REPORT to print 1 copy of the output file ACCOUNTS.LIS on queue ADM$SPOOL2.

---

43. Restrictions on a form name are necessary in the LP statement to avoid ambiguity. A form name cannot be "NO", a form name cannot contain a period (if it does it is interpreted as the output file name), and a form name cannot be V1 thru V99 (it will be interpreted as a version number specification).
44. If no form type is given, use a hyphen "-" as a placeholder before providing a file name specification to avoid ambiguity. (If no hyphen is present the fourth argument will be interpreted as a file name rather than a form name if it contains a period.)

The file name argument may be a logical name. **When REPORT encounters the file name argument it always attempts to translate it as a logical name.** If it can translate the logical name it will use the translation as the name of the output file, otherwise it will use the argument directly to name the output file, i.e. if the logical name assignment:

```
$ ASSIGN ABIGAIL.LIS MYFILE
```

was made prior to running a REPORT with the following LP statement:

```
LP 1 0 0 LETTER MYFILE
```

REPORT would name the output file "ABIGAIL.LIS" (and queue it for printing on ADM$SPOOL0 with form type LETTER).

7.17.8.4.1 Saving Multiple Versions of Output File

AdmReport allows for saving multiple versions of the output (.LIS) file on Windows when using "named" output files, e.g.

```
LP 1 0 0 NO MYREP.LIS
```

Every time a report using the LP statment is run the file MYREP.LIS is overwritten.

To avoid losing the output of previous runs, you can specify how many versions to keep (of a particular output file) by giving the number of versions in the 4th argument of the LP statement:

```
LP 1 0 0 V5 filename
```

Where V5 specifies that five versions of the output are to be kept. The V# syntax implies the NO spooling option. Valid number of versions are 1-99.

For example the statement:

```
LP 1 0 0 V3 myrep.lis
```

will result in the following files being used:

myrep[1].lis
myrep[2].lis
myrep[3].lis

When all three versions exist the file with the oldest modification date will be reused.

7.17.8.5 LP Examples

Several keywords of the LP statement may be used at the same time as is shown in the following examples:

```
LP 2 3 2     "queue 2 copies on device 3 with 2 overprints"

LP 2 3       "queue 2 copies on device 3 with no overprints"

LP 1 3 2     "queue 1 copy on device 3 with 2 overprints"

LP 2 0 2     "queue 2 copies on device 0 with 2 overprints"

LP 1 0 2 NO  "do not queue, but output 2 overprints"

LP 2 6 0 3   "queue 2 copies on device 6 with forms type 3"

LP 1 3 0 CHECKS CHECKS.LIS
             "queue 1 copy on device 3, forms type is CHECKS;
                 output file to be called CHECKS.LIS"
```

A common use of the OUTPUT and LP statements is to parameterize (see Section 7.14 "Parameterization") the options so that when the report is run a determination can be made as to where and how it is to be printed. This is illustrated in the following example:

```
OUTPUT <TI or LP?>
LP <<Copies?>> <<SPn?>>
```

7.17.9 SCALE Statement

The SCALE statement is used to scale decimal fields for printout. Created fields are not automatically scaled, but the computation may be explicitly scaled by dividing and rounding. The syntax of the statement is "SCALE N" which means to scale "N" digits, i.e. to divide and round the value by 10 raised to the N power. All values that are (sub)totals are scaled automatically, regardless if they represent the totals of actual or derived fields (built by CREATE). For example, if SCALE 3 is in effect the value 1233.34 is scaled to 1, and 14566 is scaled to 15.

SCALE NOP removes all decimal places from decimal fields, and rounds off the result. With SCALE NOP in effect 1233.34 is scaled to 1233, and 3.56 is scaled to 4.

7.17.10 PRINT ODD or EVEN

The PRINT ODD and PRINT EVEN statements are used to instruct REPORT to print only the odd numbered, or even numbered, pages of output. This feature is used to print output on both sides of the paper. First run the report with PRINT ODD. Then reload the output pages back into the printer so that the reverse side will be printed. Then run the same report with PRINT EVEN. The following example uses a substitutable parameter (see Section 7.14) to make the process even simpler:

```
REPORT SIDES
FILE BUDGET.MAS
OUTPUT KB
PRINT <Enter side to be printed, ODD or EVEN>
.
.
.
```

7.17.11 NRECS Statement

The NRECS statement instructs REPORT to only read a certain number of input records[45] and then act as if it had reached EOF. The syntax of the statement is "NRECS N" where "N" is the number of records to read.

45. If a SELECT statement is present, NRECS reads the specified number of selected records.

7.17.12 FORMAT Statement

Either one of two conditions can trigger automatic format mode as REPORT reads through the REP instruction file. Either REPORT encounters a DETAIL statement followed by field names or an asterisk "*" (see Section 7.5 “DETAIL Statement”), or REPORT encounters a FORMAT statement in the report options section.

```
FORMAT
```

The FORMAT statement should be used when any of the following conditions exist.

1. If a TABLE or LINK statement is placed before the DETAIL statement, the FORMAT statement should precede the TABLE or LINK statement.
2. If a DETAIL statement is not being used (i.e. the report produces subtotals only), the FORMAT statement must be present before the TOTAL statement is reached.

When automatic formatting is active, REPORT checks if there is a file assigned to logical name ADM$FORMAT to be used as a data description file. If this logical name is unassigned, REPORT uses default column headings and print width for the fields in the report. The use of the ADM$FORMAT data description file is described in Section 7.20 “Data Description File for Automatic Formatting”.

7.17.13 FORCE_HEADING Statement

When a KEY or SELECT statement result in no record being output, AdmReport may not output anything[46]. If output is to a file, no file would be created[47].

The FORCE_HEADING statement changes this behavior. Even when nothing would otherwise be output AdmReport will print the HEADING. This can be useful in circumstances where it is required that a file actually be created even when it contains data from no records, such as when CSV output is being generated.

7.17.14 DIRECT Statement: Multiple Output Files

REPORT has a flexible facility for directing different parts of its output to different files.

When a single report produces output which must be directed to more than one place, it is not necessary to run the report several times, or to run several different reports, or to pre-process the report file. For example (see below), a report might contain detail information, summary information on the department and division levels, and grand totals for the entire organization. The detail and totals for each department might go to various departmental managers; the department totals and division totals for each division might go to divisional managers; the division totals and grand totals might go to top management.

The DIRECT statement is used to control the destination of various sections of the output. Its syntax is as follows:

```
DIRECT SECTION_NAME QUEUE [QUEUE2 QUEUE3...]
```

SECTION_NAME: Identifies the section of the report output. SECTION_NAME is either the word 'DETAIL', or the name of one of the TOTAL fields in the report, or 'EOF', denoting the TOTAL EOF section. Because REPORT can have more than one TOTAL break on the same field, a subscript, in square brackets, identifies which TOTAL on a field is being referenced. For example, to specify that the second total statement on field EMPNO is to be queued to ADM$SPOOL5:

```
DIRECT EMPNO[2] 5
```

You need not specify '[1]' for the first TOTAL on a field.

46. Reports with at least one summary will produce output (the summaries and the heading) even when no record is selected. This output can be suppressed by putting "o" (lowercase) in the string assigned to the logical name OPTION (See Appendix A: "Options".) FORCE_HEADING will output the heading even when "o" is in OPTION.
47. Note that if the main report file contains no records, AdmReport generates no output and will not create an output file (FORCE_HEADING has no effect when the main file is empty)

QUEUE:

Between one and ten different queues can be specified, separated by blanks, using simple constants or the names of integer (I) fields[a] in the report. The values of QUEUE are between 0 and 255, corresponding to the queuing logical names ADM$SPOOL0 through ADM$SPOOL255.

When fields are used for QUEUE numbers, the output destination(s) can be changed as the report runs. REPORT might, for example, link the queue number for each department from a table keyed on the department code. If the linked queue number field is used in the DIRECT statements for DETAIL and the department TOTAL, REPORT directs those output sections to the queue number for that department.

If a QUEUE number has a value of -1, it is ignored until it is given a value between 0 and 255. This is useful in situations where you want to change the output destination "on the fly" and you may want to turn off certain output destinations. For example, you might specify four queues in a DIRECT statement, but, depending on your REPORT logic and your data, you might not always need to use all of them.

a. When CREATEd or LINKed fields are used they must appear before the DIRECT statement in which they are used.

There can be as many DIRECT statements as there are sections in a report: one for the DETAIL and one for each TOTAL. If the output of a section is not DIRECTed anywhere, REPORT uses the queue number in the LP statement, or, if there is no LP statement, it uses queue 0 (ADM$SPOOL0). A DIRECT statement may appear anywhere in a report prior to the first DETAIL or TOTAL. DIRECT statements for different sections of the report need not be in any particular order.

DIRECT provides a high degree of flexibility. Each output section can be directed to a different destination. A single section can be directed to many places. Any combination of sections can be directed to the same place. Output destinations can be controlled by logic and data when the report runs, and can be changed at any time. The output files can be queued in any desired way: all queued on the same queue, all on different queues, not queued, some queued, queued with different options, etc.

When there is at least one DIRECT statement in a report, REPORT writes a temporary LIS file (named ADMINSxxx.LIS by default or specifically named via "FILE:" in the HEADING or the LP statement) which contains the normal output and a small amount of additional control information. This temporary file is used internally by REPORT to generate the DIRECTed output files, and is not printed.

The DIRECTed output file is named by appending an underscore and the spooler number to the original LIS file name (the output files always have the extension ".LIS"):

```
Original LIS File  Spooler  Output File      Queue
-----------------  -------  ---------------  ----------
ADMINSXXX.LIS          0    ADMINSXXX_0.LIS  ADM$SPOOL0
      "                8    ADMINSXXX_8.LIS  ADM$SPOOL8
EMPLOYEE.OUT           3    EMPLOYEE_3.LIS   ADM$SPOOL3
```

The output files are sent to the ADM$SPOOLn queue corresponding to the QUEUE number.

When REPORT is finished splitting up the output, it prints a message giving the name and queue, if any, for each output file (this message is suppressed if BRIEF mode is enabled).

If for some reason REPORT terminates while splitting up the output, it is not necessary to re-run the report. When the cause of the problem is remedied, the process of splitting up the output can be restarted using the original LIS file, with the following syntax on the command line:

```
$ REPORT/MULTIPLE LIS_file_name       (OpenVMS)
 $ REPORT -MULTIPLE LIS_file_name        (Windows)
```

The "MULTIPLE" qualifier must be the first argument on the command line; and the name of the LIS file must be given. **The "MULTIPLE" qualifier is used ONLY when you want to restart the splitting process** using an existing LIS file which was produced using DIRECT statements.

With two exceptions, all REPORT features can be used in conjunction with DIRECT. Both TOTAL [PAGE] and TOTAL n (break every n records) are not compatible with DIRECT statements.

Several points about the content of DIRECT output files should be noted.

1. Each output file has its own pagination. If PGNO is used in the HEADING, correct page numbers are maintained and printed for each output file.
2. Except for the page number, the heading text in the output files is taken from the most recent heading in the original LIS file. For example, if a section starts on page 3 in the original LIS file, then, if the section begins a page in an output file, the output file will have the information from the original page 3 heading.
3. A PREVIEW and a SUMMARY at the same control break are considered to be a single section. There is no syntax for directing them to separate places.
4. Form feeds or blank lines generated by EJECT are considered to be part of the PREVIEW or SUMMARY which generates them, and they go in the same output file(s) as the PREVIEW or SUMMARY.

Several details:

1. When OUTPUT KB, TI, VT, or LA are used, REPORT checks the syntax and content of DIRECT statements, and checks the QUEUE numbers at run time, but displays the normal output and does not create any output files. This is for testing reports which have DIRECT statements.
2. If 'D' is in OPTION (see Appendix A: “Options”) and DIRECT is used, REPORT deletes the original LIS file and queues the output files to be deleted after printing.
3. To queue the output files with form types, use the form type qualifier in the ADM$SPOOLn assignments. Do not use the LP statement.
4. An output file is not queued if (1) the LP 'NO' keyword is used; or (2) the ADM$SPOOLn logical name for the file is not defined; or (3) ADM$SPOOLn does not point to a queue.
5. If a DIRECT statement contains multiple references to a QUEUE number, the duplicates are ignored.
6. No output file is created if nothing is sent to a DIRECT QUEUE.

7.17.15 DIRECT Statement Example

```
REPORT BULLISH
FILE ORG.MAS
SINGLE
HEADING
1/75
      Page PGNO-
```

```
2/1
CE    BULLISH ENTERPRISES UNLIMITED   TODAY----
BL
END
TABLE DEPTSPOOL FROM DEPTSPOOL.TAB KEY IS DEPT
TABLE DIVSPOOL FROM DIVSPOOL.TAB KEY IS DIV
*
DIRECT DETAIL DEPTSPOOL
DIRECT DEPT DEPTSPOOL DIVSPOOL
DIRECT DIV DIVSPOOL 10
DIRECT EOF 10
*
DETAIL
      PCOD- ---------THISYR  ----------LASTYR  ---------VARIANCE
END
TOTAL DEPT THISYR LASTYR VARIANCE
PREVIEW
CE   Department DEPT---
BL
END
SUMMARY
            ----------------------------------------------------
            ---------THISYR  ----------LASTYR  ---------VARIANCE
END
*
```

```
TOTAL DIV THISYR LASTYR VARIANCE
 PREVIEW
 CE   Division DIV---
 BL
 END
 SUMMARY
             ==================================================
             ---------THISYR  ----------LASTYR  ---------VARIANCE
 END
 *
 TOTAL EOF THISYR LASTYR VARIANCE
 SUMMARY
             ==================================================
             ==================================================
             ---------THISYR  ----------LASTYR  ---------VARIANCE
 END
```

7.17.16 STYLE Statement: Insert Device Control Sequences

The STYLE statement provides a means for inserting control sequences ("escape sequences") in the REPORT output. STYLE statements can be utilized for resetting the printer after use, and printing or displaying various parts of the report in different fonts or with different video attributes. This facility has the following major features:

1. It is table driven. By using different tables at run time, the same REPORT instruction file can drive different types of printers or video terminals (which use different control sequences).
2. Font changes and all other printer or video characteristics can be triggered by logic and data as the REPORT runs.
3. REPORT layout paragraphs are written in the usual way. There is no need to compensate for columns occupied by control sequences when laying out a report.

The STYLE control sequences are contained in a text editable table file identified by the logical name ADM$STYLE. There should be a different control sequence table assigned to ADM$STYLE for each output device type (because different device types have different sets of control sequences to achieve the same characteristics).

Entries in the STYLE table of control sequences are laid out as follows:

```
      .CC   MNEMONIC    CONTROL_SEQUENCE
```

".CC"	("control code") indicates that a mnemonic and a control sequence are to be found on this line (lines which do not begin with "." are ignored, and can be used for comments)
MNEMONIC	control sequence names, mnemonics made up by the user
CONTROL_SEQUENCE	a representation of the control sequence associated with the (mnemonic) name.

"Limited" support for parameterization is provided for entries in the ADM$STYLE table. For example in an entry of this form:

```
.CC INITPAGE1 MYDIR:IP1_<L_SITE>.FIL
```

AdmReport will substitute <L_SITE> with the value of the logical name L_SITE if it exists. Only "logical" parameterization is supported (no prompting ever takes place). If the logical name does not exist, or the <> parameter starts with any other characters, no substitution takes place (AdmReport assumes that the <> string is part of the control sequence).

".CC" and the mnemonic are case insensitive. Neither the mnemonic nor the control sequence can contain embedded blanks. The mnemonic string may not exceed 18 characters in length. The control sequence may be up to 255 characters in length.[48]

Nonprinting characters in the control sequence can be represented as "nnn", where "nnn" is a 1 to 3 character decimal ASCII character code between 1 and 255 (.e.g, 27 represents the escape character).[49] When this representation is used, the "nnn" counts as one character toward the limit of 18 characters in the control sequence. The value string $NULL$ in a control sequence table translates to nothing. This is for situations where some printers don't support a certain function. NULL must be all uppercase.

For example, a control sequence "REVERSE" could be defined as follows, to turn on reverse video on a VTxxx terminal:

```
.cc    reverse    $27$[7m
```

When creating control tables for different printers, the control sequence names should be standardized so that a given name translates to equivalent sequences in the different tables. For example, if the sequence for 12 characters per inch is 27[14m for an LN03 and 27[2w for a LA120, then both these sequences should be given the same name in their respective tables. This naming practice makes it possible to run the same report on different printers without changing the report instruction file (you simply assign the new table name to the logical name ADM$STYLE).

The STYLE statement identifies and defines a printing "style" consisting of one to ten control sequence names. When the report runs, the control sequences are looked up by name in the table and inserted in the report output in the order they are specified in the STYLE statement. The STYLE control sequences are inserted in the output location specified using the "S=" style designator.

The STYLE statement syntax is:

```
STYLE  style_name   up to 10 control sequence names
```

STYLE can be placed in the instruction file wherever a CREATE statement would be valid. The STYLE statement which defines a printing style must appear before the first layout paragraph where it is used. There is a limit of 29 STYLE statements in addition to SETUP, DEFAULT, and RESET.

The control sequence names can be the actual (mnemonic) names of control sequences which are in the table, and should then be enclosed in single quotes. They can also be alphanumeric (An) field names,[50] whose value (the mnemonic names to

48. A sample printer control table called PCL5.TBL, which provides some control sequences for the PCL-compatible printers, is included in the ADMINS distribution kit.
49. Non-printing characters, such as the escape character, can be "hard coded" in the table. However, the "nnn" representation is the recommended method for coding the table, because it is visible when the table is being inspected or edited. See H.2 "Integer Decimal Values for ASCII Characters" for a listing of decimal values for ASCII characters.
50. If local field names are used in a STYLE statement, the fields must be created above the STYLE statement.

be looked in the control sequence table) REPORT can change as it runs. The use of field names enables the report to change the effect of a style, i.e. to change the control sequence to be inserted, depending on logic and/or data. The following simple report displays the BALANCE field in reverse video if it is negative.

```
FILE ACCTS.MAS
OUTPUT KB
HEADING
      ACCOUNT BALANCES
END
CR NSTYLE/A8  IF BALANCE LT 0 THEN 'REVERSE' :
                 ELSE 'NORMAL' END
STYLE DEFAULT 'NORMAL'
STYLE VID NSTYLE
DETAIL
      ACCT-----
1/15
S=VID
      $$$BALANCE
END
```

A CREATE statement sets NSTYLE to 'REVERSE' when the balance is negative, and to 'NORMAL' otherwise. The STYLE statement:

```
STYLE VID NSTYLE
```

specifies that the effect of style VID is to be determined by the contents of the NSTYLE field. As the DETAIL section for each record is output, the control sequences for style VID are inserted in the location specified in the DETAIL layout (S=VID). If BALANCE is negative, REPORT looks up the mnemonic "REVERSE" (the value of the field NSTYLE) in the ADM$STYLE control sequence table, and inserts the control sequence for that mnemonic in the output. If BALANCE is positive, the control sequence with the mnemonic "NORMAL" is inserted in the output. As a result, negative balances are displayed in reverse video, others in normal video.

There are three special style names: SETUP, DEFAULT, and RESET. If a STYLE statement for SETUP is specified, its control sequences are sent once before starting to output the report. SETUP should be used for initial device setup characteristics such as margin settings or landscape mode. Similarly, if a STYLE statement for RESET is specified, its control sequences are sent once at the end of the report, to reset the device for the next user.

If a STYLE DEFAULT statement is specified, it tells REPORT what to use when no explicit STYLE is otherwise specified for a part of the output (see the example above). REPORT sets the device using the DEFAULT style once immediately before starting the report output, and also before outputting fields and literals which are laid out above the first style designator ("S=") in a layout section.

In the example above, the HEADING and the ACCT field are printed in the DEFAULT style, because no style is explicitly given for them. **STYLE DEFAULT or STYLE SETUP must be defined if any other STYLE statements are used.** If STYLE SETUP is present and STYLE DEFAULT is not, STYLE SETUP is used as the default. **Field names cannot be used in the definitions of the SETUP, DEFAULT, or RESET styles.**

The style designator "S=stylename" is placed at the left margin of the layout paragraph, (HEADING, DETAIL, PREVIEW, or SUMMARY) on a line by itself, to indicate where the output STYLE is to change, as shown in the example above. This style remains in effect until it is changed with another "S=" instruction, or until the end of the layout section.

The following points should be noted when utilizing this facility:

1. Each output line, including control sequences, can contain no more than 255 characters. The actual length of the lines varies with different control tables. REPORT checks the line length at run time and issues an error message if a line is too long to print.
2. Some device control codes, for example underlining or reverse video, change the appearance of blanks (e.g., underlined blanks, or reverse video blanks). When using such print styles, remember that while the style is set, blanks are affected.
3. When using STYLE, if a line contains any control sequences, it is not possible to "overprint" on the line (i.e., to position something to the left of what has already been laid out on the line). When using line/column precise placement in combination with STYLE, write the layout of each line in left to right order. For example, avoid laying out something on 1/50 and then, below, overprinting something else at 1/10. Instead, lay out 1/10 above 1/50. For essentially the same reason, it is not possible to use double column floating fields on lines where the print style might change.
4. When a STYLE controls the printing of information in a SUMMARY paragraph, and the STYLE statement includes one or more field names, the STYLE statement should appear under the TOTAL statement, and the fields in the STYLE statement should be referenced as fieldname/LA.

7.17.16.1 STYLE INITPAGE: Initialize Page

The reserved STYLE name INITPAGE allows you to specify a file to be dumped into the REPORT .LIS file between the formfeed and the first thing printed on each page.[51] STYLE INITPAGE is especially useful to generate a blank form before REPORT outputs data to a page. To do this, the file specified by STYLE INITPAGE would contain the printer control sequences which produce the blank form (usually a series of line graphics control sequences), ending with a "cursor control" sequence which sets the printing position back to the top of the page.

Lines in the STYLE INITPAGE file can be up to 254 characters long, and the file can contain any number of lines. This file is simply dumped, with no formatting or interpretation, into the REPORT output .LIS file at the top of each page.

The statement

```
STYLE INITPAGE style_name
```

specifies that page initialization is enabled, and each page should be initialized using the file identified in the

```
.cc style_name file_name
```

entry of the ADM$STYLE table. For example, when the statement

```
STYLE INITPAGE W2_STATE
```

appears, REPORT will look for an entry in the ADM$STYLE table such as:

```
.cc w2_state treas_forms:w2_state.repform
```

The above entry would cause REPORT to dump the contents of file[52] "TREAS_FORMS:W2_STATE.REPFORM" into the .LIS file at the beginning of each page.

---

51. On the first page the SETUP or DEFAULT style control sequence always immediately follows the formfeed.

STYLE INITPAGE should not be used with DIRECT (multiple output files, as described in Section 7.17.14 "DIRECT Statement: Multiple Output Files") or with arbitrary length text fields (see Section 7.6.1 "Text Fields") which may generate extra page breaks.

7.17.16.1.1 Using STYLE INITPAGE to Place An Image on Each Page of an ADMINS Report Output

AdmReport provides a way to include images (such as logos or pictures) in the blank form generated for every page of report output.

Lets say you have an image of your company logo or the official seal of your city or town that you want to place on each page of report output. The image is likely in one of the more common image formats, with a file type extension of BMP, JPG, GIF or TIF. To use this image with STYLE INITPAGE it must first be converted to PCL format.[53]

Lets assume our file is called cityseal.pcl and we have placed it in a directory or folder referenced by the logical name MYFILES.

Our goal is to produce pages of AdmReport output that contain both a picture (logo, seal etc.) and the normal AdmReport output. If the PCL file produced by HiJaak ends with a form feed, the image would print by itself on a new page after every page of the AdmReport output. What we need is to change the PCL file to eliminate the unwanted form feed character, and to reset the cursor position back to the top of the page so that AdmReport starts its output in the usual place on the page (otherwise it would start the output at whatever position it was in after rendering the image). One way (the hard way) to accomplish this is to manually edit the file, remove unwanted form feeds, and insert the control sequences necessary to put the cursor back at the top of the page. This technique can be successful, but some text editors might change the file's format and contents in ways that make the image unusable (especially on OpenVMS systems). A better and far easier way is to use the ADMINS INITPAGE utility.

The INITPAGE utility is designed specifically to alter PCL files so that they can be used with AdmReport's STYLE INITPAGE capability. If you call INITPAGE without any arguments it displays the following:

```
>initpage
       The INITPAGE utility is used to add cursor positioning
       to a file containing PCL code to be used by the INITPAGE
       mechanism in ADMINS REPORT.
usage: initpage FILENAME ROW COLUMN
```

INITPAGE takes three arguments. The first is the name of the file to be altered. The second and third arguments are the row and column where the cursor is to placed after the INITPAGE file is output. When INITPAGE runs, it eliminates any final form feed from the file it is acting on, and then inserts the control sequences necessary to put the cursor at the specified location on the page. In the example below the file myfiles:cityseal.pcl would be altered so that the cursor position would be set to row 1 column 0 (where AdmReport output starts on a page) after the contents cityseal.pcl are printed.

```
>initpage myfiles:cityseal.pcl 1 0
```

52. Any logical name in the file specification is translated only once, before REPORT begins printing.
53. ADMINS has used HiJaak from Imsisoft to perform this conversion.

Place an entry in the ADM$STYLE table file for the pcl version of the image.

For cityseal.pcl in the MYFILES directory that entry would look like this:

```
.cc seal myfiles:cityseal.pcl
```

where "seal" is the mnemonic name that will be used to reference this entry in the STYLE INITPAGE statement of the report instruction file (.REP). The entry for cityseal.pcl would look like this:

```
STYLE INITPAGE SEAL
```

When this report is run, AdmReport uses the file you have specified to "initialize" each page, then overlays that page with the normal AdmReport output.

Example: Put "The Unified Community" logo in report output

1. Convert the file uclogo.gif into a PCL file with HiJaak. Here's what uclogo.gif looks like:

2. Then run INITPAGE on uclogo.pcl so it works with AdmReport.

```
initpage uclogo.pcl 1 0
```

3. Put an entry in the ADM$STYLE table that references uclogo.pcl.. For this example we'll use test.tbl as the ADM$STYLE table. Here's the contents of test.tbl:

```
.cc SEAL uclogo.pcl

.cc RESET  $27$E
```

4. Place the STYLE INITPAGE command in the report instruction file referencing uclogo.pcl via its mnemonic "seal". Here's a sample report instruction file with the STYLE INITPAGE command inserted.

```
FILE MYFILES:MYDATA.MAS-R
AUTO
SINGLE
OUTPUT LP
LP 0 0 0
WIDTH 80
STYLE INITPAGE seal
HEADING
1/15
     Report: TEST1
       Page: PGNO-
1/40
     The Unified Community
     PRINT STYLE TESTING
1/64
     Date: TODAY----
     Time: NOW--
BL
BL
BL
BL
END
DETAIL
       DMSG-------------------------------------------------
bl
```

7.18 TOTAL Paragraph Options

Just as there are options which apply to the entire report, there are options which may apply to each TOTAL paragraph. The statements are placed in the TOTAL paragraph following the TOTAL statement. Any of the options may be used in any or all of the TOTAL paragraphs as desired.

7.18.1 SUPPRESS Statement

Generally speaking a subtotal summarizes several or many detail lines. In some instances, however, there may only be one detail line contributing to a subtotal. Usually when there is only one detail line in a subtotal the information provided by the SUMMARY paragraph and the information provided by the DETAIL paragraph to the reader of the report are the same, i.e. one of the two may be redundant.

The SUPPRESS statement is used to instruct REPORT not to print (i.e. to suppress) the SUMMARY paragraph whenever only one line of detail occurred in the run preceding the subtotal. For example, if in a budget printout we were printing object-of-expense lines as our detail and subtotaling by class of object then those classes that only contained one instance could be printed without the class subtotal line by using the SUPPRESS statement. The SUPPRESS statement is placed between the TOTAL which REPORT is to "suppress" and the SUMMARY paragraph.

An option in the SUPPRESS statement is "SUPPRESS ZERO". "SUPPRESS ZERO" will not print the SUMMARY paragraph for total breaks where a "break field" of the type "Xpicture" is a value of zero. This can be significant for financial reporting systems with many levels of account coding where some transactions are not coded at the minor levels and totals are not desired for the transactions without codes.

SUMMARY sections can be suppressed conditionally by giving SUPPRESS a Boolean criterion. For example:

```
TOTAL ORG DEPTOT/V
SUPPRESS DEPTOT/V EQ 0
SUMMARY
```

When the SUPPRESS expression is true, the SUMMARY is not printed. As in a CREATE under a TOTAL, the SUPPRESS expression should use explicit aggregate field names (like DEPTOT/V above). There can be only one conditional SUPPRESS per TOTAL; but a conditional SUPPRESS can be combined with a SUPPRESS or SUPPRESS ZERO statement in the same TOTAL section.

7.18.2 EJECT Statement

The EJECT statement tells REPORT to start a new page (i.e. "eject" the current page) after printing a SUMMARY paragraph. The EJECT is placed between the TOTAL statement and the SUMMARY statement of the subtotal control break at which it is to eject. There can be an EJECT for each TOTAL statement.

The EJECT BEFORE statement tells REPORT to start a new page **before** printing the SUMMARY paragraph.

The EJECT AFTER statement tells REPORT to start a new page **after** printing this SUMMARY paragraph, or else **after printing any other SUMMARY paragraphs that are pending** and will print immediately below this SUMMARY.

The EJECT n statement instructs REPORT to start a new page if, after printing the SUMMARY paragraph, n or less lines remain on the page.

7.18.3 RESET PAGE statement

The RESET PAGE statement is used after a TOTAL statement to reset the internal page counter PGNO (see Section 7.16 "Internal Field Names") after the control break, as follows:

```
TOTAL PARCEL field1 field2 ...
RESET PAGE 2
```

In this example the page counter PGNO would be reset to "2" after the control break.

If no value is given on the line after "RESET PAGE" PGNO will be reset to "1".

7.19 EXECUTE Statement: RMO Processing

REPORT may call an RMO at any number of processing points using the EXECUTE statement.[54] Reports which are logically complex or require looping calculations can be streamlined by using an RMO to eliminate pre-processing steps, or to write the logic in a more compact way than is possible using the standard "GOTO-less" REPORT syntax. The RMO is fully compatible with other REPORT features.

EXECUTE statements call the RMO, which is named in the file statement in lieu of the main REPORT file (the main REPORT file is the file the RMO was compiled for). An EXECUTE statement can appear at any point in a report where a CREATE statement would be valid. Therefore, the RMO can be called at whatever processing points are desired: before a LINK, after a LINK, under a TOTAL, before a SELECT or a SORT, etc.

Optionally, any string of up to 6 characters can follow the EXECUTE verb. If it is present, this string is placed in the optional special RMO local field S$S/A6 when REPORT calls the RMO, so that different parts of the RMO logic can be executed at different calls, as in TRANS. (If there is only one EXECUTE statement in a report, then there is no need to use S$S in the RMO.) If there is no S$S string after the EXECUTE verb, REPORT calls the RMO with a blank value in S$S if S$S is a local field.

54. An RMO is a Record Maintenance Procedure. See Chapter 9: "CMP: The Record Maintenance Compiler" for details on RMO syntax and preparation.

For example:

```
* RMO.REP
*
FILE STUFF.RMO
*
EXECUTE PRE
*
LINK XN FROM M.MAS KEY IS M
*
EXECUTE POST
*
DETAIL
     XN---------
END
```

In the example, at each record REPORT calls the RMO with 'PRE' in S$S, then performs the LINK, then calls the RMO again with 'POST' in S$S, and finally prints XN.

LINKed, TABLEd, and CREATEd fields can be used in the RMO if they are declared as local RMO fields. Local RMO fields which are not declared in REPORT are available to REPORT, as though they were fields in the main file. For example:

```
* STUFF.REP
*
FILE STUFF.RMO
EXECUTE PRE
LINK XM FROM M.MAS KEY IS MPRE
EXECUTE POST
TOTAL EOF MPOST/LA
SUMMARY
      MPOST/LA----
END

* STUFF.RMS
*
FILE N.MAS
LOCAL
S$S/A6
MPRE/I
XM/I
MPOST/I
PROGRAM
IF S$S EQ 'PRE' THEN MPRE = M + 100 ; STOP ; END
IF S$S EQ 'POST' THEN MPOST = XM / 2 ; STOP ; END
*
```

In this example, the RMO calculates the link field MPRE in the 'PRE' call. After the link executes, the RMO uses the linked field XM to calculate MPOST, which is totaled. Note that MPRE and MPOST are declared only in the RMO but can be used freely in the REPORT.

To invoke RMO test mode, place the qualifier "TEST" on the REPORT command line. The "TEST" qualifier can be used in conjunction with any other REPORT command line arguments, in any position.

The only special fields currently supported in REPORT RMOs are SS, QQ, and TODAY. As before, Q$Q should only be set in the detail section of a report (that is, not beneath a TOTAL statement). P$P can be used to print values from the RMO when testing a report; but P$P should not be used in production reports because P$P produces output lines which are not counted by REPORT.

Elements of local arrays used in the RMO must be referenced in the REPORT via an ALIAS statement in the RMO, as described in Section 9.5.3 “ALIAS: Create Field Names for Local Array Elements”.

REPORT opens files "-R" (read-only) by default. When an RMO is used in REPORT, the data file is opened "-R" unless there is an explicit file open mode in the RMS FILE statement (for example, FILE PHOGG.MAS-X).[55]

The initial setup and "compilation" phases of REPORT take slightly longer when an RMO is used. Once that is finished, the execution speed of REPORT is the same, whether one uses an RMO or the equivalent CREATE statements. The advantages of using an RMO lie not in simply replacing CREATE logic with RMO logic, but rather in streamlining the logic and eliminating pre-processing steps.

Note 1: Using Subtotals in the RMO

Subtotaled fields can be used in the RMO, but not directly. A field such as N/MAX cannot be declared as a local field in the RMS. Instead, the value of the subtotaled field is placed in a CREATEd field, and the CREATEd field is declared as a local field in the RMO. For example:

```
* POOP.REP
FILE POOP.RMO
    ...
TOTAL N N/MAX N/MIN
CR RMID/I 1
CR RMAX/I N/MAX
CR RMIN/I N/MIN
EXECUTE TOT
SUMMARY
      RMID---------
END

* POOP.RMS
FILE N.MAS
LOCAL
S$S/A6
RMAX/I
RMIN/I
RMID/I
PROGRAM
IF S$S EQ 'TOT' THEN RMID = RMIN + ((RMAX - RMIN) / 2) ;
  STOP ; END
```

Note 2: Using RMO with SORT

When there is a SORT statement in the REPORT, each EXECUTE statement causes RMO calls either before or after the SORT is performed. EXECUTE statements before a SORT statement call the RMO before the SORT is performed, but not afterwards. Likewise, EXECUTE statements which follow a SORT statement call the RMO only in the post-sort pass.

55. Alternative file open modes in REPORT are used to extend or restrict concurrent access to the file by other users while the REPORT is running. REPORT never writes to ADMINS data files.

7.19.1 REP$SECLEN - Controlling Section Length in the RMO

When REPORT finishes printing an output section (a DETAIL, SUMMARY, etc.), it makes a decision about whether to start a new page before printing the next section. This prevents REPORT from splitting a section across pages, from printing a preview as the last thing on a page, and so forth.

When making these decisions, REPORT assumes that the next section(s) to be printed will contain as many output lines as are specified in the layout for the section(s).

It often happens, though, that because of floating fields and the suppression of blank fields, the actual number of lines printed is less than the number of lines in the layout. Usually, the difference is small, and makes little or no difference in how the output is paginated.

But a small number of reports have sections whose length varies greatly, which can cause REPORT to start a new page much earlier than it needs to, leaving an excessive number of blank lines at the end of the page. A special field, REP$SECLEN, can be used to tell REPORT how many lines a DETAIL or SUMMARY section will contain. This advance information lets REPORT make better decisions about pagination in reports with highly variable length sections.

REP$SECLEN/I can only be used with an RMO (see Section 7.19 "EXECUTE Statement: RMO Processing"), and should be declared as a local field in the RMS. To implement this facility, determine how many lines should be allowed for the current record's DETAIL section in the RMO, and set REP$SECLEN to this number.

If REP$SECLEN is set by an RMO call (EXECUTE statement, see Section 7.19 "EXECUTE Statement: RMO Processing") that occurs **after a TOTAL statement,** then REP$SECLEN specifies the number of lines to be allowed for that TOTAL's SUMMARY section.

REP$SECLEN can be used only with DETAIL and SUMMARY sections: it has no effect on HEADINGs or PREVIEWs.

REP$SECLEN is automatically set to -1 before each RMO call. If the RMO does not set it to some value between 0 and the number of lines in the layout section, REP$SECLEN has no effect, and the REPORT is paginated based on the length of the layout section.

REP$SECLEN is a specialized feature which should be used only when necessary, for unusual reports where the developer needs to have control over pagination. It requires an RMO, and the RMO must contain logic which calculates the number of lines in one or more layout sections as the report is running.

In some cases, REP$SECLEN may be useful in conjunction with the LAYOUT feature (see Section 7.13.10 "Layout Statement: Variable Formatting"). Using LAYOUT, a single layout paragraph can contain several different layouts, some of which may be considerably longer than others. REP$SECLEN can be used to set the length of the layout which will actually be used to print a given record or summary.

A general point about calling the RMO under a TOTAL: when the RMO is called under a TOTAL, REPORT has already read the next record in the file (otherwise, it would have no way to know that it's at the end of a control break). All fields available to the RMO except S$S and CREATEs under the TOTAL will have values reflecting the first record in the next control break.

If you call the RMO under a TOTAL, and you need information from fields which have already been re-set by the first record of the next control break, you may be able to keep it in local fields in the RMS; but there is an easier way which involves less RMO programming and is much less confusing.

For example, assume there is a field N in the file, and you need the value of N for the last record in the control break in an RMO call under a TOTAL. If it's not already there, add N/LA to the TOTAL line, then CREATE a field under the total to hold this value, as follows:

```
.
.
.
TOTAL KEYFLD 1DATA 2DATA ... N/LA
*
CREATE NTOT/I N/LA
EXECUTE SETLEN
*
SUMMARY
```

The CREATE statement must be before the EXECUTE statement.

If NTOT/I is then declared in the local section, the RMO will have the value from N/LA when its called by the EXECUTE statement.[56]

7.20 Data Description File for Automatic Formatting

When automatic formatting as described in Section 7.5 "DETAIL Statement" is active, REPORT automatically builds column headings. These column headings can be prepared in a data description file and used in place of the field names that REPORT would otherwise use by default. To use a data description file during automatic formatting, assign a file name to the logical name ADM$FORMAT.

The DEF of a file which can be used as a data description file for REPORT, should include the following required fields.

```
* ADMFORMAT.DEF
* Required fields in data description file
*
MAS 100
FILE An  KEY1     "File Name"
FIELD An KEY2     "Field Name"
HEAD1 An          "Heading, line 1"
HEAD2 An          "Heading, line 2"
HEAD3 An          "Heading, line 3"
WIDTH I           "Column width"
NT I              "Non-totalling"
```

As field names are encountered by REPORT when it is doing automatic formatting, REPORT checks the file assigned to the logical name ADM$FORMAT to see if these field names are present for the particular report master file. If they are present, REPORT uses the headings, column width, and non-totaling status information from ADM$FORMAT.

---

56. The CREATE statement is necessary because you cannot declare a field named "N/LA" in the RMO.

REPORT searches for fields in ADM$FORMAT under the report master file name, identified in the FILE statement in the REP instruction file. Created fields are searched for under this file name as well. Fields in link or table files are searched for under the link or table file name. REPORT will not know to look for link or table fields in the data description file unless automatic formatting has been activated prior to the LINK or TABLE statements. Since LINK and TABLE statements usually occur before a DETAIL statement, a REP instruction file which is to use automatic formatting should include a FORMAT statement before the LINK or TABLE is encountered.

Non-totaling status tells REPORT whether or not to automatically total numeric fields. Numeric fields (decimal and four word decimal field types) are automatically totaled only when the NT field equals zero.

The data description file may have other fields which will be ignored by REPORT.

In the following example, the file DATADESC.MAS which has been defined as above, is assigned to the logical name ADM$FORMAT. Once ADM$FORMAT is assigned REPORT will use it for retrieving information about the fields in a report.

```
$ assign datadesc.mas adm$format
```

To disable REPORT from using DATADESC.MAS as the data description file, the logical name ADM$FORMAT should be deassigned.

7.20.1 Example Using a Data Description File

The file DEPTEMP.MAS has three fields, Department Number (DEPT/KEY1), Employee Number (EMP#/KEY2), and Salary (SALARY). The file EMPLOYEE.MAS has a key of EMP# and many fields including the Employee First and Last Name, FIRST and LAST. DEPTNAME.TAB is a table of Department Numbers and Names. DNAME is the Department Name.

The following data description file records are included in a file DATADESC.MAS.

```
FILE          FIELD    HEAD1       HEAD2    HEAD3   WIDTH NT

DEPTEMP.MAS   DEPT     DEPT        #                4     1
DEPTEMP.MAS   EMP#     ID#                          8     1
DEPTEMP.MAS   NAME     EMPLOYEE    NAME             24    1
DEPTEMP.MAS   SALARY   SALARY      PER      YEAR    10    0
DEPTNAME.TAB  DNAME    DEPT        NAME             9     1
EMPLOYEE.MAS  EMP#     EMPLOYEE    NUMBER           8     1
EMPLOYEE.MAS  FIRST    FIRST       NAME             12    1
EMPLOYEE.MAS  LAST     LAST        NAME             12    1
```

The following report with automatic formatting will be produced first using the default automatic format, and then using the format information in the data description file, DATADESC.MAS.

```
REPORT SALARIES
FILE DEPTEMP.MAS
FORMAT
TABLE DNAME FROM DEPTNAME.TAB KEY IS DEPT
LINK FIRST LAST FROM EMPLOYEE.MAS KEY IS EMP#
CREATE BLANK/A1 ' '
CREATE NAME/A24 NCAT(NAME,FIRST,BLANK,LAST)
DETAIL DEPT EMP# NAME SALARY
TOTAL DEPT DNAME/FI
TOTAL EOF
```

Running the report prior to assigning the data description file to ADM$FORMAT results in the following:

```
DE DNAME/FI  EMP# NAME                              SALARY

12           1021 Audrey Allen                    23,000.00
12           1254 Cornelius Cosmos                34,500.00
12           9256 Bruce Short                     27,000.00
                                                  ----------
12 Personnel                                      84,500.00
...
...
                                                  ----------
                                                  277,000.00
```

Note that field names are used for the heading within the default field widths.

By assigning DATADESC.MAS to the logical name ADM$FORMAT, the following report is produced using the same REP instruction file above.

```
DEPT DEPT      ID#  EMPLOYEE                 SALARY
#    NAME           NAME                     PER
                                             YEAR

12             1021 Audrey Allen              23,000.00
12             1254 Cornelius Cosmos          34,500.00
12             9256 Bruce Short               27,000.00
                                             ----------
12   Personnel                                84,500.00
...
...
                                             ----------
                                             277,000.00
```

The column headings and widths from the data description file are used. Note that the created field NAME has been included in the description file as one of the DEPTEMP.MAS field names. The non-totaling field, NT in DATADESC.MAS, has been set for no totaling in the non-numeric fields; but in this example these fields would not be totaled anyway because they are alphanumeric and integer fields.

7.21 Pre-Compiled Reports

REPORT normally compiles the .REP instruction form each time a report is run. REPORT can alternatively compile a report in an executable RPO file some time before the report is actually run, and then execute the pre-compiled RPO file at run time. Pre-compiled reports have two main benefits. (1) Complex reports which run on a small number of records take significantly less time to run, since the compilation phase is not repeated every time the report runs. (2) End users can run pre-compiled reports without having access to REPORT source instructions, which is desirable in some environments.

To compile a report without executing it, use the following command line syntax:

```
$ REPORT  REP_file_name [REPORT_NAME] /COMPILE  (OpenVMS)
$  REPORT  REP_file_name [REPORT_NAME] -COMPILE  (Windows)
```

REPORT_NAME is optional, and is needed only when the .REP file contains more than one report. The "COMPILE" qualifier must be the last argument on the command line, and there can only be two or three arguments.

When /COMPILE is specified, REPORT creates an executable .RPO file. If a REPORT_NAME is specified, then REPORT_NAME.RPO is created; otherwise, the .RPO file has the same first name as the .REP file, as in the following examples:

```
$ REPORT TESTREP REPORT3/COMPILE   creates REPORT3.RPO

$ REPORT TESTREP/COMPILE           creates TESTREP.RPO
```

If no REPORT_NAME is specified, the .RPO file is created in the same directory as the .REP file. If a REPORT_NAME is used, the .RPO file is created in the user's default directory. However, if the logical name ADM$OBJECT is defined to point to a directory, then the .RPO file is always placed in the ADM$OBJECT directory.

To run a report using a .RPO file, the command line syntax is:

```
$ REPORT RPO_NAME/RPO       (OpenVMS)
 $ REPORT RPO_NAME -RPO        (Windows)
```

The RPO_NAME is the first name of the .RPO file: **do not** include the ".RPO" extension. There can only be two command line arguments and the "RPO" qualifier must be the second argument.

When a report is run from a .RPO file, REPORT checks to make sure that the field names and types, and the order of the fields in the DEFs of all the files are the same as they were when the report was compiled. If the report uses an RMO, REPORT checks to see whether local fields in the RMO have been added, changed, rearranged, etc. If any of the file DEFs or the RMO local fields have changed since the .RPO was created, REPORT exits with a message and the report must be recompiled.

Values for substitutable parameters (see Section 7.14 “Parameterization”) are supplied during the compilation phase. They are stored in the .RPO file. When the report is run using the .RPO file there is no prompting for substitutable parameters. If it is necessary to supply a parameter to a pre-compiled report at run time, the parameter value can be assigned to a logical name. The report can then use the TRLOG subroutine to obtain the value of the parameter at run time. Unlike <parameters>, this method of passing information to a pre-compiled report is limited to passing the values of fields.

7.22 The REPORT Environment File

AdmReport has an environment file which can be used to customize its behavior, designed to work in the same way that the TRANS environment file modifies TRANS behavior (see Section 6.17 “The TRANS Environment File”). AdmReport uses the logical name REPORT_ENV to find the Report Environment File.

AdmReport supports the following keyords in the Report Environment File

keyword	function
dmap	has the same purpose and syntax as the dmap statement in TRANS$ENV: it remaps output characters so they display appropriately in a different character set than the character set they in which they are stored (see Section 6.17.4 "DMAP and MAP" for a complete discussion of dmap syntax). If dmap is used, all REPORT output is remapped: literals as well as data
option	Provides a way for AdmReport to "add or subtract" from the options specified in the logical name OPTION. For example: **`option+0`** ensures that "Option 0" behavior (display zero values in numeric fields as blanks) is specified, whether or not it is present in the logical name OPTION at run time, while: **`option-0`** ensures that "Option 0" behavior is not specified. You may also use this method to locally enable or disable option "P" (use parentheses to indicate negative numbers) and option "," (comma: supress billions, millions, thousands delimiter in numeric fields).
progressbar	Tells report to display an activity indicator (or "progress bar") showing that the report is currently still processing. Syntax: progressbar=x,y where x and y are pixel values for whereyou want the progress bar to appear on the desktop (0,0 is upper left corner), or: progressbar=center to center the progress bar on the desktop.
exceloptions	Rename the file typeof AdmReport -Excel output. Syntax: **`Exceloptions=ftype=XXX`** Where XXX e.g. is "xls" (or ".xls") to name the AdmReport -Excel output .xls instead of .xml. (This would prevent "-XML" and "-EXCEL" output from being produced with the same name (and thus one overwriting the other), and also allow Office 2002 to correctly launch Excel when double-clicking the file (Office 2003 correctly launches Excel when "XML" files are output using the "-EXCEL" option).

7.23 Report Overlay Feature (MERGE)

The MERGE facility makes it possible to merge the results of several reports into one output printout. MERGE can often be used to simplify what would be a complex reporting operation by splitting it into two or more simpler reports, and then merging their outputs.

Report overlay files are specified by placing the MERGE statement:[57]

```
MERGE:merge-file-name
```

by itself on the first line of the HEADING of the report.

After all REPORTs that are contributing files to be merged are run there will exist text files with the names "merge-file-name.DAT;1", "merge-file-name.DAT;2", etc., in the user's file directory. These files are overlaid, or "merged", by the MERGE command.

```
$ merge merge-file-name ti/lp [r]
```

Output can be to the terminal or the printer. If the output is to the printer, the output will be queued to the device assigned to ADM$SPOOL0, as described under the spooling system (see Chapter 21: "Printer Queues"). The presence of the "R" instructs MERGE **not** to delete the input .DAT files. (If "R" is absent, MERGE deletes these files after it has created the output report.) "R" would be used if the user wishes to run MERGE repeatedly on these files, or the user wishes to use the files for some other purpose after running MERGE. Note, however, that the user must see to it that these files are deleted before running the REPORTs again to recreate these .DAT files.

57. Windows systems do not support file version numbers, so the developer must explicitly identify the "version" to be created by each REPORT in the MERGE statement for that REPORT (e.g. MERGE:CALENDAR.DAT;1 in the REPORT that creates the first file for merging, and MERGE:CALENDAR.DAT;2 in the REPORT that creates the second file for merging, etc.).

7.24 Report Command Line Options

AdmReport supports command line options that allow you to create report output in various formats and/or view report output via various applications.

7.24.1 View: DIsplay output via TedRE /V

Use AdmReport's "/view" option to display the resulting .LIS file using TedRE's special read-only "view" mode. In this special TedRE mode, if you opt to print the report it will spool the original .LIS file in the same way AdmReport does.

This option allows the user to view the report output (with "printer control language" sequences filtered out), while still retaining the capability to print the report exactly as AdmReport would have printed it directly, that is, with the printer control language sequences included.

7.24.2 HTML: Create an HTML "wrapper" for output.

AdmReport can be run with the **/HTML** switch to place an HTML wrapper around the output from REPORT. This allows the output file to be viewed as a preformatted document in a browser. The syntax is:

```
AdmReport /HTML[=filename] [/browse] REPNAME
```

The output file name has the file type **.htm** instead of the standard **.lis**. If no filename follows the **/html** switch, a standard report output file name with the file type **.htm** is created, unless a file name is specified by the LP statement or a FILE: statement in the report source file, in which case this file name will be used.

If the **/BROWSE** switch is present on the command line AdmReport for Windows starts the browser to view the output file.

The HTML tags generated by the /HTML switch are:

```
<html>
<head>
<style type="text/css">
@media print
{
pre.first {}
pre.notfirst {page-break-before: always}
</style>
</head>
<body>
<pre class="first">

Output from Report is inserted here

</pre>
</body>
</html>
```

Most of the above HTML is there to support correct pagination of the report output when it is printed directly from the broswer. AdmReport insert the following HTML tags at each point in the output where ordinary AdmReport output would start a new page:

```
</pre><pre class="notfirst">
```

The result is HTML output that displays correctly[58] in the browser and prints correctly when using the browser's print command.

7.24.2.1 ADM_HTML_STYLEn: Customized HTML wrapper

You can specify the HTML tages used to wrap the output from REPORT. Create a file with the following generalized format:

```
HTML tags/statements to be output before the output from REPORT
<!--REPORT_PAGEBREAK ccccc-->
<!--REPORT_OUTPUT-->
HTML tags/statements to be output after the output from REPORT
```

The line **<!--REPORT_OUTPUT-->** is replaced by the output from REPORT.

If you are concerned about proper pagination of this output when printed from the browser use the **<!--REPORT PAGEBREAK ccccc -->** token to specify what HTML content to insert (**ccccc** represents the content to be inserted) at each point in the output where ordinary AdmReport would start a new page[59].

Use this command line syntax:

```
AdmReport /HTMLn[=filename] [/browse] REPNAME
```

where ***n*** is any positive number or zero and is used to designate which logical name in the form:

ADM_HTML_STYLEn

points to the "HTML wrapper" file you want to be used

7.24.2.2 Example: Customized HTML wrapper

The following customized HTML wrapper file would cause the report output HTML file to print in landscape mode, and paginate correctly, if printed from the browser.

```
<html>
<head>
<style type="text/css">
@media print
{
pre.first {writing-mode: tb-rl}
pre.notfirst {page-break-before: always;writing-mode: tb-rl}
</style>
</head>
<body>
<pre class="first">
<!--REPORT_PAGEBREAK </pre><pre class="notfirst">-->
<!--REPORT_OUTPUT-->
</pre>
</body>
</html>
```

---

58. STYLE statements that cause, for example, PCL control sequences to be embedded in the report output should not be used with the HTML command line switch.The browser does not recognize them as control sequences so it will attempt to display them on the screen and in printed output.
59. The "**<!--REPORT_PAGEBREAK ccccc-->**" line, if used, must appear before the "**<!--REPORT_OUTPUT-->**" line. The trailing "-->" is mandatory, and there must be a space following the "**<!--REPORT_PAGEBREAK** "before the HTML content to output, as shown in the example above.

Lets say this wrapper file is named "landscape_wrapper.htm" and resides in a folder identified by the logical name "myfiles". We make the following logical name assignemnt:

```
admlcr adm_html_style8 myfiles:landscape_wrapper.htm
```

and then call report referencing our customized wrapper file:

```
admreport -html8 myreport
```

The output is "HTML-wrapped" using the customized wrapper file!

7.24.3 CSV: read special CSV instructions in the ".REP" file

AdmReport's -CSV command line "switch" allows you to write specific code for CSV output in the same report that contains code for standard reports and/or XML output. This enables standard reports, XML reports, and CSV reports to use the same virtual record.

The command line syntax is:

```
AdmReport -CSV
AdmReport -CSV=outputfile
```

The .REP must contain standard CSV report lines starting with:

```
*!csv!
```

(See Section 7.5.2 "DETAIL *CSV: Output in CSV Format" and Section 7.10.3 "Summary *CSV statement : CSV output based on TOTALs" for details on how to specify CSV output from AdmReport.)

These lines will be seen as comments if the report is run without the -CSV switch. The report instructions to be used when executed with the -CSV switch must immediately follow the *!CSV! prfix. E.g.:

```
*!csv!HEADING
*!csv!     "ACCOUNT","TEXT","AMOUNT"
*!csv!END
*!csv!DETAIL *CSV ACCOUNT TEXT AMOUNT
```

When the "-CSV" command line option is used, all formatting instruction (DETAIL, TOTAL, SUMMARY etc) sections are discarded, while the virtual record remains intact).

7.24.4 Excel: Create output in Excel's XML format

Use AdmReport's -EXCEL command line option to create output in Microsoft Excel XML format[60]

60.'You'll need Microsoft Office 2002 or later to use Microsoft Excel XML format. This XML format is also "recognized" by Windows as an Excel file. OpenOffice Calc can also handle Microsoft Excel XML format. To specify that the ***browse*** option should launch Calc instead of Excel use the -CALC command line option instead of -EXCEL, or (with -EXCEL on the command line) include the following in the REPORT Environment File (REPORT$ENV)

```
excel_browse=OpenOffice.org Calc
```

XML for Excel is created from ordinary .REP syntax, and a single report instruction file (.REP) can be used to create standard REPORT (.LIS) output, and Excel XML output. Because Excel is, after all, a spreadsheet, this capability is best suited for report layouts where the data are neatly aligned in columns.

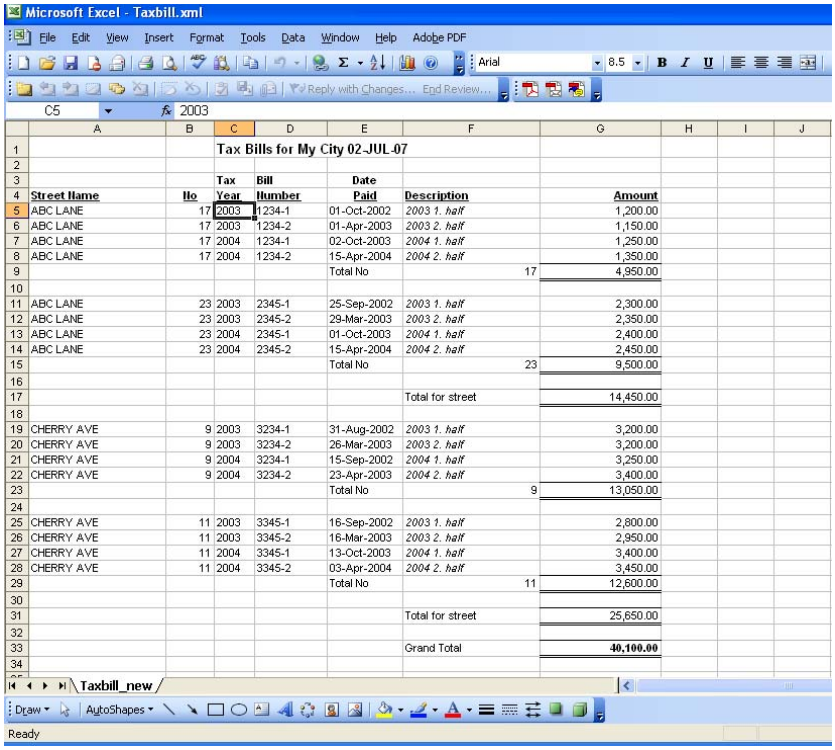
The following simple report:

```
report taxbill
file RMOSUB:taxbill.mas
single
lp 1 0 0 no Taxbill.txt
select amount ne 0
*!Excel!style 70 parent=45 format=37
heading
                          Tax Bills for My City today----
bl
                            Tax   Bill     Date
     Street Name         No Year  Number   Paid     Description               Amount
bl
end
detail
     street---------- ---no yea- bill#--- date----- descr----------- ---------amount
end
*!Excel!column 6 style=22
total no amount
summary
                                          Total No  no---            ---------amount
bl
end
*!Excel!column 7 style=45
total street amount
summary
                                                    Total for street ---------amount
bl
end
*!Excel!column 7 style=45
total eof amount
*!Excel!ignore line=1
*!Excel!ignore line=3
*!Excel!column 7,2 style=70
summary
                                                                      ---------------
                                                    Grand Total       ---------amount
                                                                      ===============
end
```

could create the following output:

```
                      Tax Bills for My City 03-JUL-07

                         Tax  Bill      Date
Street Name         No Year Number    Paid     Description                 Amount

ABC LANE            17 2003 1234-1   01-OCT-02 2003 1. half              1,200.00
ABC LANE            17 2003 1234-2   01-APR-03 2003 2. half              1,150.00
ABC LANE            17 2004 1234-1   02-OCT-03 2004 1. half              1,250.00
ABC LANE            17 2004 1234-2   15-APR-04 2004 2. half              1,350.00
                                     Total No  17                        4,950.00

ABC LANE            23 2003 2345-1   25-SEP-02 2003 1. half              2,300.00
ABC LANE            23 2003 2345-2   29-MAR-03 2003 2. half              2,350.00
ABC LANE            23 2004 2345-1   01-OCT-03 2004 1. half              2,400.00
ABC LANE            23 2004 2345-2   15-APR-04 2004 2. half              2,450.00
                                     Total No  23                        9,500.00

                                               Total for street         14,450.00

CHERRY AVE           9 2003 3234-1   31-AUG-02 2003 1. half              3,200.00
CHERRY AVE           9 2003 3234-2   26-MAR-03 2003 2. half              3,200.00
CHERRY AVE           9 2004 3234-1   15-SEP-02 2004 1. half              3,250.00
CHERRY AVE           9 2004 3234-2   23-APR-03 2004 2. half              3,400.00
                                     Total No  9                         13,050.00

CHERRY AVE          11 2003 3345-1   16-SEP-02 2003 1. half              2,800.00
CHERRY AVE          11 2003 3345-2   16-MAR-03 2003 2. half              2,950.00
CHERRY AVE          11 2004 3345-1   13-OCT-03 2004 1. half              3,400.00
CHERRY AVE          11 2004 3345-2   03-APR-04 2004 2. half              3,450.00
                                     Total No  11                        12,600.00

                                               Total for street         25,650.00

                                                                     ---------------
                                               Grand Total                40,100.00
                                                                     ===============
```

Run with the /Excel switch you would get Taxbill.xml which in Excel would show:



Microsoft Excel - Taxbill.xml

Tax Bills for My City 02-JUL-07

	Street Name	No	Tax Year	Bill Number	Date Paid	Description	Amount
5	ABC LANE	17	2003	1234-1	01-Oct-2002	*2003 1. half*	1,200.00
6	ABC LANE	17	2003	1234-2	01-Apr-2003	*2003 2. half*	1,150.00
7	ABC LANE	17	2004	1234-1	02-Oct-2003	*2004 1. half*	1,250.00
8	ABC LANE	17	2004	1234-2	15-Apr-2004	*2004 2. half*	1,350.00
9					Total No	17	4,950.00
11	ABC LANE	23	2003	2345-1	25-Sep-2002	*2003 1. half*	2,300.00
12	ABC LANE	23	2003	2345-2	29-Mar-2003	*2003 2. half*	2,350.00
13	ABC LANE	23	2004	2345-1	01-Oct-2003	*2004 1. half*	2,400.00
14	ABC LANE	23	2004	2345-2	15-Apr-2004	*2004 2. half*	2,450.00
15					Total No	23	9,500.00
17						Total for street	14,450.00
19	CHERRY AVE	9	2003	3234-1	31-Aug-2002	*2003 1. half*	3,200.00
20	CHERRY AVE	9	2003	3234-2	26-Mar-2003	*2003 2. half*	3,200.00
21	CHERRY AVE	9	2004	3234-1	15-Sep-2002	*2004 1. half*	3,250.00
22	CHERRY AVE	9	2004	3234-2	23-Apr-2003	*2004 2. half*	3,400.00
23					Total No	9	13,050.00
25	CHERRY AVE	11	2003	3345-1	16-Sep-2002	*2003 1. half*	2,800.00
26	CHERRY AVE	11	2003	3345-2	16-Mar-2003	*2003 2. half*	2,950.00
27	CHERRY AVE	11	2004	3345-1	13-Oct-2003	*2004 1. half*	3,400.00
28	CHERRY AVE	11	2004	3345-2	03-Apr-2004	*2004 2. half*	3,450.00
29					Total No	11	12,600.00
31						Total for street	25,650.00
33						Grand Total	**40,100.00**

The lines starting with ***!Excel!*** are seen as comments when the report is run in "normal" mode (creating and/or printing a ".LIS" file), but when run with the ***/Excel*** command line switch, these lines are interpreted as instructions to AdmReport's Excel module that will modify its behavior. The report would create an Excel spreadsheet without any *!Excel! modifiers, but you would then get default values for everything.

The syntax for creating an Excel XML file is:

```
AdmReport /Excel[=filename] [/browse] reportname
```

The /Excel switch may be followed by a file name. If no file name is given the output file will be named via AdmReport's standard methods, but the file type will always be XML.[61] , for example "admins0000561.xml". If run with the /browse switch AdmReport will launch Excel once the output file is produced. The */Excel* switch may be followed by a file name, if none is supplied AdmReport will create a name for the output file name automatically The file type will always be modified to .XML. If run with the */browse* switch AdmReport will bring up Excel once the .XML file is produced.

61. The latest versions of Microsoft Office and Microsoft Windows will recognize the XML file as being formatted for Microsoft Excel. For earlier versions, that do not recognize the XML file as an Excel file, use the following statement in the REPORT$ENV file to tell AdmReport to instead use the file type XLS, which will be recognized as an Excel file.

    Exceloptions=ftype=XLS

If a report is run with the /Excel command line switch it will ignore option K (which - for use in Europe - reverses the use of "," and "." for thousand separator and decimal point). Also, if option E is present AdmReport uses "A4" as the Excel output spreadsheet's print page size instead of "Letter" size).

When AdmReport calculates the width and data types of the columns it will use the layout of the *DETAIL* section if there is one. If no *DETAIL* section is present it will use the first *SUMMARY* section.

It is very important that the output data in all sections are properly aligned and sized, as all output will be directed to the column where its first printable character falls. As an example, look at the 7th column in our simple report above, where we specify the column size of the *AMOUNT* field. If we in the *DETAIL* section had specified:

```
-----AMOUNT
```

and in a *SUMMARY* section had specified:

```
-------AMOUNT
```

the *SUMMARY* section amount would have appeared in column 6, since its first (possible) printable character would fall within the boundaries of column 6 as calculated by the *DETAIL* section.

As for the *HEADING* section, the last and possibly the next to last, heading line will be tried to be used as column headers provided all text (or data) falls within the column boundaries as calculated from the *DETAIL* (or *SUMMARY*) section. If any printable character crosses a column boundary the line is rejected as a column header. If your column header text is wider than the data item in the *DETAIL* section, stretch out the space for the data item with dashes. E.g. if you want to print a three digit FUND number in the *DETAIL* section and want the word ***Fund*** to appear as a column heading, specify ***FU--*** not ***FU-*** in the *DETAIL* section.

If a column header's first character overlaps the first position of a column the column header is left justified. If the last character of column header overlaps the last position of the column, and there are no character overlapping the first position the column header is right justified. In all other cases the column header is centered.

When report would output a page break (Form Feed), a Page Break instruction for printing purposes is inserted into the Excel XML output. The heading section will be repeated at the top of each page printed from Excel, so if the report is printed from Excel you will get the same page breaks as you would if the report was run in "normal" mode to a printer. Alternatively, use the instruction

```
*!Excel!pagebreak columnheaders=only
```

to tell AdmReport to display (and repeat at the top of each page of printed output) only the column headers it recognizes from the heading layout, rather than the entire heading.

The appearance of the data in a column may be modified by using ***!Excel!column*** statements. These statements must follow the layout section they are modifying, and apply to that layout section only. In the the simple .REP at the beginning of this document ***!Excel!column*** statements follow the DETAIL and the TOTAL/ SUMMARY sections (for a TOTAL/SUMMARY section the *!Excel!column may be anywhere after the TOTAL statement as long as it appears before the next TOTAL statement).

The syntax of the ***!Excel!column*** instruction is:

```
*!Excel!column column#[,line#] keyword=value
```

Observe that the column number always refers to the columns as defined by the DETAIL section (or the first SUMMARY section if no DETAIL section is present). The "***,line#***" is only necessary if there is more than one line in the current section, and you are not referring to a column on line 1.

Currently the only implemented keyword is ***style***. To modify the style to use for a column, use e.g.:

```
*!Excel!column 6 style=22
```

to have column 6 appear left justified in *italic*.

7.24.4.1 Styles Available.

The default font is 8.5 point Arial. There are a number of styles provided to control how data (and literals) are presented. Styles can specify the font, the point size, whether to use bold, italic or underline (or any combination of the three), whether to right justify, center or left justify the text, and how to format and present numbers and dates.

Styles are identified by a number. Currently the following styles are available when outputting numeric fields:

10	Basic number displayed as 1,234 (decimal places in report output data are rounded to whole number)
11	Basic number rounded to 2 decimal places
12	Basic number with 2 decimal places, negative values in parentheses
13	Currency symbol, 2 decimal places
14	Currency symbol, 2 decimal places, negative values in parentheses

Value from Report	Display via... Style 10	Style 11	Style 12	Style 13	Style 14
1234	1,234	1,234.00	1,234.00	$ 1,234.00	$ 1,234.00
1234.5	1,235	1,234.50	1,234.50	$ 1,234.50	$ 1,234.50
-117.506	-118	-117.51	(117.51)	$ -117.51	$ (117.51)

Styles 20 – 43 deal with various combinations of alignment and highlighting, and will usually be used for text data:

Highlighting	Aligned Left	Aligned Center	Aligned Right
Plain	20	28	36
Bold	21	29	37
Italic	22	30	38
Bold Italic	23	31	39
Underline	24	32	40
Bold Underline	25	33	41
Italic Underline	26	34	42
Bold Italic Underline	27	35	43

Some other "special purpose" styles:

44	**Bold**, left justified, point size 10
45	Same as 11, but with a single border line on top, and a double border line underneath.
46	Like 45, but based on 12
47	Like 45, but based on 13
48	Like 45, but based on 14
49	Like 45, but based on 10

Styles 45-49 can be combined with styles 20-43 for alignment and highlighting. E.g.

```
*!Excel!style 70 parent=49 format=37
```

will create style 70 which displays numbers as whole numbers (no decimals), right alligned, bolded with a single line border on top and a double line border at the bottom.

Styles 50- 52 are used for date formats. Using September 3, 2007 as an example, the formats are:

Style Code	Description	Example
50	Date	03-Sep-2007
51	Short date	9/3/2007
52	Medium date	3-Sep-07

The default date style is ***50***. The developer may choose one of the other two built-in styles by using the ***!Excel!datestyle*=ss**, e.g.

```
*!Excel!datestyle=51
```

to choose style ***51*** for all date fields.

Developers may also modify the default date style ***50*** to format the date any way they want by using the ***!Excel!dateformat=***format instruction, e.g.

```
*!Excel!dateformat=d-mm-yyyy
```

to format the date as ***3-09-2005***.

The characters used to create the format mask are:

d	1 or 2 digit day.
dd	always use 2 digits for day (i.e. 03, not just 3)
m	1 or 2 digit month number
mm	always use 2 digits for month
mmm	Use 3 characters for month name (e.g. Jan, Feb etc.)
mmmm	Use full month name
yy	2 digit year
yyyy	4 digit year

In addition, almost any character may be used as separator characters, e.g. ***yyyy-mm-dd*** or ***mm/dd/yyyy***. To make sure that the separator character always is treated as a literal character to be inserted, precede the character with a \ (backslash), e.g. ***dd\/mm\-yyyy***.

Instead of having to specify the ***datestyle*** or the ***dateformat*** statements in the report source files they may be put in the ***REPORT$ENV*** file and thus take effect in all reports. The ***REPORT$***ENV syntax is:

```
Exceloptions=datestyle=ss
Exceloptions=dateformat=format
```

7.24.4.2 User Defined Styles.

The styles defined above basically fall into two categories:

Styles that describes a data type (e.g. style 12)
and
Styles that describes some highlighting and alignment options (styles 20-43).

Developers can define their own styles by combining styles from these two categories, to specify number styles with highlighting (note: alignment options are ignored as number styles are always right justified). The syntax is:

```
*!Excel!style ## parent=## format=##
```

E.g.

```
*!Excel!style 70 parent=12 format=37
```

would create style number 70 that would display a number with two decimals right aligned in bold.

User defined styles must be numbered from 70 and up. Parent styles must be in the range 10-13 and formats in the range 20-43. User defined style statements can be placed anywhere in the .REP instruction file, and the styles defined can be referenced in DETAIL and at all levels of TOTAL.

7.24.4.3 Page Printing .

Excel reports have page printing setup capability. This feature is controlled by the **PageSetup** keyword, in Excel reports coded as:

```
*!Excel!PageSetup
```

By default Excel Reports print

```
Page # of N printed date at time
```

PageSetup controls page headers and footers which will only show up on printed output (they can be viewed when the Excel window is open, but do not appear as part of the general display).

Following the *!Excel!PageSetup keyword are any number of lines describing what headers and footers to print. The general syntax is:

```
*!Excel!<pagepart>_<just> Text to print
```

where **<pagepart>** is either Header or Footer (controlling whether the text will be printed at the top or at the bottom of the page), and **<just>** is L, C or R controlling whether the text is right or left justified, or centered (you may think of the header and footer sections as having three panels, left, center and right, where each panel is controlled independent of the other panels).

An example of using the Page setup feature would be:

```
*!Excel!PageSetup
*!Excel!Header_L Report %%REPNAME%%
*!Excel!Header_C %%LONGNAME%%
*!Excel!Header_C For <%L_CUSTOMER>
*!Excel!Header_C Created <%TODAY> at <%NOW>
*!Excel!Header_R Page &amp;P of &amp;N
*!Excel!Footer_C Printed &amp;D at &amp;T
```

Field names from the virtual record are enclosed in double %-signs. <%L_name> will be replaced by the value of the logical name L_name. <%TODAY> and <%NOW> will be replaced by the current date and time when the Excel report was created. In addition you may use a number of internal Excel keywords to access certain features at print time. Currently you may use:

```
&amp;PCurrent page number
&amp;NTotal number of pages
&amp;DCurrent date
&amp;TCurrent time
```

The example above might print something like:

Report Taxbills Preliminary Tax Bills Page 1 of 6

For City of High Hopes

Created 11/13/2007 at 2:13 PM

Excel spreadsheet body ....

....

Printd 11/13/2007 at 6:45 PM

provided, of course, that the various L_name logical names have properly assigned values when the report is run.

As can be seen from the example above, each new occurence of a reference to a panel will start a new line in that panel. In our example there are three *!Excel!Header_C lines, which will create three lines in the center header panel.

The different lines for a panel need not be coded next to each other. If we e.g. wanted to add a second line to the right header panel it could coded after the *!Excel!Footer_C line above, making it possible to define your "standard" heading in an include file, and then add individual lines to each panel after the include file, either in another include file, or hard coded in the .REP.

7.24.4.4 Miscellaneous Keywords.

The ***!Excel!*** has to appear in front of all the following:

`ignore line=#`	Ignore (do not include) line number '#' in this layout section (in our example we use it to get rid of lines 1 and 3 in the TOTAL EOF SUMMARY section (Instead we use "user defined style" 70).
`layout landscape`	Use landscape mode when printing from Excel.
`gridlines`	Print gridlines when printing from Excel

7.24.5 XML: create XML output

ADMINS now has a new "-xml" option that functions with AdmReport. This "-xml" switch changes REP's behavior so that it reads special XML preprocessor commands in the .REP instruction file and generates a valid XML document as output.

The -xml option can be called in three ways:

```
admreport -xml demo
admreport -xml=special demo
admreport -xml=SO
```

In the first case above -xml appears by itself. Use this construct to have the resulting xml file take its name from the .REP file that is specified, so in this case demo.xml is generated.

Use the second construct, -xml=special, to explicitly name the file produced, e.g. "special.xml" would be created.

Use -xml=SO to specify direct ouput to standard output, no xml file to be created. If any other string besides "SO" occurs after "-xml=" that string will be used to name the xml file produced.

7.24.5.1 Generation of XML

The idea is that you can write new .REP files that contain special XML preprocessing instructions (lines that begin with "*!xml!", "*!xmltotal!" or "*!xmlmore!"), or insert these special commands in existing REPORTs to take advantage of "virtual records" that have already been assembled.

Here's a new .REP written expressly for XML-production:

```
*demo.rep
*
```

```
REPORT VOTERSTREET
FILE VOTER.MAS-R
LINK PLACE FROM POLLING.TAB KEY IS DIST
CREATE XSN/I CCAT(XSN,#STR)
CREATE ASSEM_DIST/X999 ASSEM
SORT ASSEM STREET #STR
*!xml! #vot name xsn street #apt party dist assem
*!xmltotal! street assem_dist/FI #vot/E
*!xmltotal! assem #vot/E
```

The Report XML preprocessor converts this input into the following .REP, which is given to REPORT.

```
REPORT VOTERSTREET
FILE VOTER.MAS-R
SINGLE
WIDTH 254
LENGTH 0
OUTPUT LP
LINK PLACE FROM POLLING.TAB KEY IS DIST
CREATE XSN/I CCAT(XSN,#STR)
CREATE ASSEM_DIST/X999 ASSEM
SORT ASSEM STREET #STR
HEADING
     FILE:demo.xml
     «?xml version=ø1.0ø encoding=øiso-8859-1ø?»
     «?xml-stylesheet type=øtext/xslø href=øc:\bills\conf2004\demos\admdefault4lw.xslø?»
END
DETAIL
     «record»
      «no_vot» #vot------------------------------------------------------------------------ «/no_vot»
      «name» name-------------------------------------------------------------------------- «/name»
      «xsn» xsn---------------------------------------------------------------------------- «/xsn»
      «street» street---------------------------------------------------------------------- «/street»
      «no_apt» #apt------------------------------------------------------------------------ «/no_apt»
      «party» party------------------------------------------------------------------------ «/party»
      «dist» dist-------------------------------------------------------------------------- «/dist»
      «assem» assem------------------------------------------------------------------------ «/assem»
      «/record»
END
TOTAL street assem_dist/FI #vot/E
PREVIEW
     «summary_1»
END
SUMMARY
      «street_total_1» street-------------------------------------------------------------- «/street_total_1»
      «assem_dist_fi_1» assem_dist/FI------------------------------------------------------ «/assem_dist_fi_1»
      «no_vot_e_1» #vot/E------------------------------------------------------------------ «/no_vot_e_1»
      «/summary_1»
END
TOTAL assem #vot/E
PREVIEW
     «summary_2»
END
SUMMARY
      «assem_total_2» assem---------------------------------------------------------------- «/assem_total_2»
      «no_vot_e_2» #vot/E------------------------------------------------------------------ «/no_vot_e_2»
      «/summary_2»
END
TOTAL EOF
PREVIEW
     «!DOCTYPE DOCUMENT â «!ELEMENT DOCUMENT
     ((record¶summary_1¶summary_2¶summary_3¶summary_4¶summary_5¶summary_6¶summary_7¶summary_8¶summary_9)*)»
     «!ELEMENT record (no_vot,name,xsn,street,no_apt,party,dist,assem)»
     «!ELEMENT summary_1 (record*,street_total_1,assem_dist_fi_1,no_vot_e_1)»
     «!ELEMENT summary_2 (summary_1*,assem_total_2,no_vot_e_2)»
     «!ELEMENT no_vot (#PCDATA)» «!ELEMENT name (#PCDATA)» «!ELEMENT xsn (#PCDATA)» «!ELEMENT street (#PCDATA)»
     «!ELEMENT no_apt (#PCDATA)» «!ELEMENT party (#PCDATA)» «!ELEMENT dist (#PCDATA)» «!ELEMENT assem (#PCDATA)»
     «!ELEMENT street_total_1 (#PCDATA)» «!ELEMENT assem_dist_fi_1 (#PCDATA)» «!ELEMENT no_vot_e_1 (#PCDATA)»
     «!ELEMENT assem_total_2 (#PCDATA)» «!ELEMENT no_vot_e_2 (#PCDATA)»
     ô»
     «DOCUMENT»
END
SUMMARY
     «/DOCUMENT»
END
```

Here are some representative snippets from the output when this .REP is run:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="x:\admins\bin\admdefault4.xsl"?>
<!DOCTYPE DOCUMENT [ <!ELEMENT DOCUMENT
((record|summary_1|summary_2|summary_3|summary_4|summary_5|summary_6|summary_7|summary_8|summary_9)*)>
<!ELEMENT record (no_vot,name,xsn,street,no_apt,party,dist,assem)>
<!ELEMENT summary_1 (record*,street_total_1,assem_dist_fi_1,no_vot_e_1)>
<!ELEMENT summary_2 (summary_1*,assem_total_2,no_vot_e_2)>
<!ELEMENT no_vot (#PCDATA)> <!ELEMENT name (#PCDATA)> <!ELEMENT xsn (#PCDATA)>
<!ELEMENT street (#PCDATA)> <!ELEMENT no_apt (#PCDATA)>
<!ELEMENT party (#PCDATA)> <!ELEMENT dist (#PCDATA)> <!ELEMENT assem (#PCDATA)>
<!ELEMENT street_total_1 (#PCDATA)> <!ELEMENT assem_dist_fi_1 (#PCDATA)> <!ELEMENT no_vot_e_1 (#PCDATA)>
<!ELEMENT assem_total_2 (#PCDATA)> <!ELEMENT no_vot_e_2 (#PCDATA)>
<!ATTLIST DOCUMENT repinfo_date CDATA #FIXED "12-Feb-2010">
<!ATTLIST DOCUMENT repinfo_time CDATA #FIXED "13:56:22">
<!ATTLIST DOCUMENT repinfo_who CDATA #FIXED "BD"> ]>
<DOCUMENT>
<summary_2>
<summary_1>
<record>
 <no_vot> 018493</no_vot>
 <name> MANDERVILLE, MARY L</name>
 <xsn> 15</xsn>
 <street> BEACON VIEW DRIVE</street>
 <no_apt></no_apt>
 <party> D</party>
 <dist> 05</dist>
 <assem> 127</assem>
```

```
 </record>
<record>
 <no_vot> 018491</no_vot>
 <name> MANDERVILLE, CHARLES E JR</name>
 <xsn> 17</xsn>
 <street> BEACON VIEW DRIVE</street>
 <no_apt></no_apt>
 <party> U</party>
 <dist> 05</dist>
 <assem> 127</assem>
 </record>
<record>
 <no_vot> 018492</no_vot>
 <name> MANDERVILLE, HELEN M</name>
 <xsn> 17</xsn>
 <street> BEACON VIEW DRIVE</street>
 <no_apt></no_apt>
 <party> R</party>
 <dist> 05</dist>
 <assem> 127</assem>
 </record>
<record>
 <no_vot> 019323</no_vot>
 <name> MCEVILY, TODD M</name>
 <xsn> 21</xsn>
 <street> BEACON VIEW DRIVE</street>
 <no_apt></no_apt>
 <party> U</party>
 <dist> 05</dist>
 <assem> 127</assem>
 </record>
<record>
 <no_vot> 025816</no_vot>
 <name> RUTKA, BARBARA A</name>
 <xsn> 58</xsn>
 <street> BEACON VIEW DRIVE</street>
 <no_apt></no_apt>
 <party> U</party>
 <dist> 05</dist>
 <assem> 127</assem>
 </record>
<record>
 <no_vot> 020654</no_vot>
 <name> MORAWSKI, LILLIAN</name>
 <xsn> 68</xsn>
 <street> BEACON VIEW DRIVE</street>
 <no_apt></no_apt>
 <party> U</party>
 <dist> 05</dist>
 <assem> 127</assem>
 </record>
 <street_total_1> BEACON VIEW DRIVE</street_total_1>
 <assem_dist_fi_1> 127</assem_dist_fi_1>
 <no_vot_e_1> 6</no_vot_e_1>
 </summary_1>
.
.
.
<record>
 <no_vot> 018678</no_vot>
 <name> MARMOR, HARRIS</name>
 <xsn> 178</xsn>
 <street> WYNN WOOD DRIVE</street>
 <no_apt></no_apt>
 <party> D</party>
 <dist> 07</dist>
 <assem> 134</assem>
 </record>
 <street_total_1> WYNN WOOD DRIVE</street_total_1>
 <assem_dist_fi_1> 134</assem_dist_fi_1>
 <no_vot_e_1> 3</no_vot_e_1>
 </summary_1>
 <assem_total_2> 134</assem_total_2>
 <no_vot_e_2> 1,438</no_vot_e_2>
 </summary_2>
</DOCUMENT>
```

The following shows a typical ADMINS .REP file to which XML preprocessor commands have been added:

```
REPORT RESALE
FILE RE.MAS-R
SINGLE
OUTPUT <<Enter Print Device TI or LP>>
LP 1 0 0 RESALE.LIS
*
nrecs 500
LINK NAME FROM REACCOUNT.MAS KEY IS 1ACCT
*
CO BL/A1
CO TLOT/A6 NCAT(TLOT,LOT,BL,EXT)
*
```

```
SELECT PURCHASE GT 0
*
SORT LSTREET LHSE
*
STYLE SETUP LANDSCAPE 13_CPI SKIP_P LINESINCH8
*
HEADING
1/1
     ***RESALE***
1/45
     REAL ESTATE SALES REPORT
1/90
     TODAY----
1/120
     PAGE: PGNO-
BL
3/1
     Owner's Name              Sale Date   Vol  Page   Acct#
3/60
     Property Location      Map Lot    Unt   Assessment  Sale Price
BL
END
*
DETAIL
1/1
C    NAME--------------------- ACQDA----  -VOL -VPAG   1ACCT-
1/60
   LHSE- LSTREET---------- MA- TLOT-- UNT- ------GROSS ---PURCHASE
END
*
*!xml! name acqda vol vpag 1acct lhse lstreet
*!xml! ma tlot unt gross purchase
*!xmltotal! lstreet gross/max gross/min purchase/max purchase/min
TOTAL EOF MAP/E
SUMMARY
BL
     *** Grand Total Parcels: MAP/E---
END
*
```

Note that the preprocessor commands look "commented", so that if this .REP were given to REPORT without using the "-xml" option the preprocessor commands would be ignored.

7.24.5.2 REPORT's XML Preprocessor

REPORT's XML Preprocessor works as follows.

If REPORT is called with the "-xml" option, all lines in the input file are ignored except the lines that begin with any of the following strings starting in column 1:

Report Statements

FILE

LINK

TABLE

CR **(create)**

CO **(compute)**

EX **(execute)**

NREC

SEL **(select)**

ORSEL **(orselect)**

KEY

SORT

Any of the above lines that are encountered **before an XML preprocessor command** are passed to the output .REP without change. Once an XML preprocessor command is encountered, only preprocessor commands and CREATE/COMPUTE commands are recognized and processed.

XML Preprocessor Statements

*!XMLSTYLESHEET!	see "L$XSL_STYLESHEET: Specifying the stylesheet" below
*!XMLOPTIONS!	
*!XML!	XML preprocessor "DETAIL" command
*!XMLTOTAL!	XML preprocessor "TOTAL" command
*!XMLMORE!	XML preprocessor "TOTAL" continuation command

Once an XML preprocessor command is encountered only XML preprocessor commands (and CReate or COmpute statements that occur after an XMLTOTAL) are processed (all other subsequent REPORT syntax is ignored).

7.24.5.3 XMLOPTIONS Statement

The XMLOptions statement is used to specify alternate behaviors. The following alternate behaviors are supported:

Keyword	Behavior
NOCOMMA	Suppresses commas in decimal and integer fields.

Currently only "nocomma" is supported. Keywords are specified in a blank-separated list, as follows:

```
*!xmloptions! keyword1 keyword2 keyword3
```

7.24.5.4 XML Statement

The XML statement specifies an ADMINS Report DETAIL section to the preprocessor.

Multiple XML statements are allowed but they must be consecutive lines (no intervening lines of any kind).

Multiple XML statements result in a single DETAIL section in the report instruction file output by the preprocessor.

XML statements encountered after an XMLTOTAL statement are ignored.

Each time a DETAIL section is output an XML "record" element is created, with each of the fields specified in the XML statement output as "child" elements of the "record" element.

Fields from the virtual record identified in the XML statement are output as XML text elements with the field name (all lowercase) used as the tag, unless the field name is an invalid tag. If the field name begins with a number the tag for the text element will have the lead number replaced with the number spelled out, followed by an underscore, e.g. ADMINS field "0FLD" will have a tag of "zero_fld" and ADMINS field "2ADDR" will have a tag of "two_addr". If the field name includes the pound sign, "#", the string "no_" is substituted for the pound sign in the tag for that element (thus ADMINS field name VOTER#1 becomes XML tag voterno_1).

As we saw in the example above the XML statement:

```
*!xml! #vot name xsn street #apt party dist assem
```

results in the following DETAIL section when output from the preprocessor:

```
«no_vot» #vot------------------------------------------------------------ «/no_vot»
«name» name-------------------------------------------------------------- «/name»
«xsn» xsn---------------------------------------------------------------- «/xsn»
«street» street---------------------------------------------------------- «/street»
«no_apt» #apt------------------------------------------------------------ «/no_apt»
«party» party------------------------------------------------------------ «/party»
«dist» dist-------------------------------------------------------------- «/dist»
«assem» assem------------------------------------------------------------ «/assem»
«/record»
```

This DETAIL section, in turn, results in XML like the following being output for each record when the report is run:

```
<record>
 <no_vot> 018493</no_vot>
 <name> MANDERVILLE, MARY L</name>
 <xsn> 15</xsn>
 <street> BEACON VIEW DRIVE</street>
 <no_apt></no_apt>
 <party> D</party>
 <dist> 05</dist>
 <assem> 127</assem>
 </record>
```

7.24.5.5 XMLTOTAL and XMLMORE Statements

The XMLTOTAL statement specifies an ADMINS Report TOTAL statement and SUMMARY section to the preprocessor. XMLMORE statements provide continuation lines for the XMLTOTAL statements that they follow. Each group of XMLTOTAL and any subsequent XMLMORE statements specify a single TOTAL/SUMMARY grouping.

The item after the *!xmltotal! token specifies the break field for the report TOTAL statement. It may be either:

- a **field name** to specify a control break whenever that field changes (must be a key field or SORT field)
- **EOF** to specify a TOTAL EOF
- **A number** to specify that the report should output a SUMMARY whenever that number of records is output. ***NOT YET IMPLEMENTED***

Each time an XMLTOTAL preprocessor section is encountered an XML "summary_n" element is created (where n is the summary number, first XMLTOTAL/XMLMORE group creates is summary_1, second creates summary_2, etc.), with each of the fields specified in the XMLTOTAL/XMLMORE statements output as "child" elements of the "summary_n" element.

All record elements (resulting from XML statements as described above) are "children" of summary_1 elements if XMLTOTAL occurs. In turn all summary_1 elements are children of summary_2 elements if they exist, and so on for each succeeding higher level of summary created by each succeeding XMLTOTAL section encountered. Record and summary_n child elements occur before the child elements resulting from the fields specified in the XMLTOTAL/XMLMORE statements.

Thus xml preprocessor statements in this form

```
*!xml!  fields…
*!xmltotal!  "total" fields…
*!xmltotal!  "total" fields…
*!xmltotal!  "total" fields…
```

produce a xml document with the following generalized structure

```
<summary_3>
    <summary_2>
        <summary_1>
            <record> (field elements here) </record>
             one record element for each "DETAIL" record
            <record> (field elements here) </record>
            (summary_1 "total" field elements here)
        </summary_1>
        one summary_1 element for each 1st level TOTAL break
        <summary_1>
             one record element for each "DETAIL" record
            (summary_1 "total" field elements here)
        </summary_1>
        (summary_2 "total" field elements here)
    </summary_2>
     one summary_2 element for each 2nd level TOTAL break
    (summary_3 "total" field elements here)
<summary_3>
one summary_3 element for each 3rd level TOTAL break
```

If a field name is specified for a break the string "_total_" followed by the summary number is appended to the tag for that field. For example, in the following XMLTOTAL line:

```
*!xmltotal! assem #vot/E amt 1addr
```

ASSEM is the total break field. If this line were the second XMLTOTAL encountered then the tag for ASSEM would be "assem_total_2". Other (non-break) fields from the virtual record identified in the XMLTOTAL/XMLMORE statements that do not have aggregation operators will be output in the same way. In the example above the field

AMT would be output with the tag "amt_total_2". As explained above for the XML (DETAIL) statement, if the tag that would be generated would be invalid because the field name begins with a number or contains the "#" character, a valid tag is substituted. Thus in the example above field 1ADDR would be output with tag "one_addr_total_2". When Adm Report's aggregation operators are appended to the field name, e.g. #VOT/E above, valid tags are constructed by replacing the "/" with "_" and appending another "_" followed by the summary number. Thus #VOT/E is given the tag "no_vot_e_2".

Thus the XMLTOTAL statement:

```
*!xmltotal! street assem_dist/FI #vot/E
```

results in the following TOTAL statement, PREVIEW section and SUMMARY section when output from the preprocessor:

```
TOTAL street assem_dist/FI #vot/E
PREVIEW
     «summary_1»
END
SUMMARY
      «street_total_1» street------------------------------------------------------ «/street_total_1»
      «assem_dist_fi_1» assem_dist/FI---------------------------------------------- «/assem_dist_fi_1»
      «no_vot_e_1» #vot/E---------------------------------------------------------- «/no_vot_e_1»
      «/summary_1»
END
```

This TOTAL statement and SUMMARY section, in turn, results in XML like the following being output for each control break when the report is run:

```
 <summary_1>
.
.
.
 <street_total_1> WYNN WOOD DRIVE</street_total_1>
 <assem_dist_fi_1> 134</assem_dist_fi_1>
 <no_vot_e_1> 3</no_vot_e_1>
 </summary_1>
```

7.24.5.6 XML Attributes

XML "element attributes" can be specified in the XML Preprocessor commands, as in the following example:

```
REPORT RESALE
FILE RE.MAS-R
SINGLE
LINK NAME FROM REACCOUNT.MAS KEY IS 1ACCT
*
CREATE BL/A1
CREATE TLOT/A6 NCAT(TLOT,LOT,BL,EXT)
CREATE COLOR/A10 IF PURCHASE GT 500000 THEN 'YELLOW' ELSE ' ' END
CREATE FONT/A2 IF ACQDATE GT 'April 1, 1998' THEN '16' ELSE ' ' END
*
```

```
SELECT ACQDATE BET 'January 1, 1998' AND 'December 31, 1998'
SELECT PURCHASE GT 0
*
SORT LSTREET LHSE
*
*!xml! name(desc="Buyer") acqdate(desc="Date of Sale") vol vpag
1acct lhse lstreet
*!xml! ma tlot unt gross(desc="Assessed Value" aln="R")
*!xml! purchase(desc="Purchase Price" aln="R" color?=red font?=12)
*!xmltotal! lstreet(desc="summary for" aln=C) gross/max(desc="high
valuation" aln=R)
*!xmlmore! gross/min(desc="low valuation" aln=R ) purchase/
max(desc="high sale" aln=R)
*!xmlmore! purchase/min(desc="low sale" aln=R sumbgcolor?=white
sumfgcolor?=teal)
CREATE SUMBGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE
' ' END
CREATE SUMFGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE '
' END
*!xmltotal! eof  gross/max(desc="high valuation" aln=R)
*!xmlmore!  gross/min(desc="low valuation" aln=R ) purchase/
max(desc="high sale" aln=R)
*!xmlmore!  purchase/min(desc="low sale" aln=R eofbgcolor?=white
eoffgcolor?=teal)
CREATE EOFBGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE
' ' END
CREATE EOFFGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE '
' END
```

In the XML Preprocessor statements note the contents of the parentheses, for example in the line:

```
*!xml! name(desc="Buyer") acqdate(desc="Date of Sale") vol vpag
1acct lhse lstreet
```

the grouping

name(desc="Buyer")

tells the preprocessor that the XML child element "name" should have the attribute "desc" with a default value of "Buyer", while the grouping

acqdate(desc="Date of Sale")

tells the preprocessor that the XML child element "acqdate" should have the attribute "desc" with a default value of "Date of Sale".

Default attributes for child elements (fields in ADMINS Report) of the record elements (DETAIL sections) or the summary_n elements (SUMMARY sections) are implemented by outputting a an XML Document Type Definition (DTD) that specifies the default value for the attribute of the element. The entries in the DTD generated by the "name" and "acqdate" groupings discussed above would look like this:

```
<!ELEMENT name (#PCDATA)>
<!ELEMENT acqdate (#PCDATA)>
.
```

```
.
<!ATTLIST name desc CDATA "Buyer">
<!ATTLIST acqdate desc CDATA "Date of Sale">
```

Attributes can be determined dynamically by tying the attribute value to a field in the report virtual record with the REPORT XML Preprocessor "?=" operator:

```
*!xml! purchase(desc="Purchase Price" aln="R" color?=red font?=12)
```

This line tells the XML preprocessor that the element "purchase" has four attributes: "desc" with a default value of "Purchase Price"; "aln" default value "R"; "color" default value "red"; and "font" default value "12". In addition, the "?=" operator used for the color and font attributes means that the ADMINS virtual record will have fields named COLOR and FONT and if either of these fields have a non-null value then that value should be output for that attribute for that instance of the purchase element. It is the developers responsibility to make sure the fields referenced with the "?=" operator are present in the report virtual record. The sample report instruction file above includes these the two create statements:

```
CREATE COLOR/A10 IF PURCHASE GT 500000 THEN 'YELLOW' ELSE ' ' END
CREATE FONT/A2 IF ACQDATE GT 'April 1, 1998' THEN '16' ELSE ' ' END
```

The combination of these CREATE statements and the use of the "?=" operator results in the purchase element being output with the color attribute set to "YELLOW" if PURCHASE has a value greater than 500,000 (and set to its default of "RED" otherwise); and with the FONT attribute set to "16" if ACQDATE is after "April 1, 1998" (and set to its default of "12" otherwise).

The following shows what the resultant XML would look like for the purchase element with the COLOR and FONT attributes being set dynamically:

```
<purchase
  color = " YELLOW                                                    "
  font = " 16                                                         "
 > 775,000</purchase>
```

The XML preprocessor supports CREATE statements after XMTOTAL/XMLMORE groupings to facilitate the use of dynamically set attributes for child elements of summary_n elements. The following lines, excerpted from the above sample report instruction file, illustrate this capability:

```
*!xmltotal! lstreet(desc="summary for" aln=C) gross/max(desc="high valuation" aln=R)
*!xmlmore! gross/min(desc="low valuation" aln=R ) purchase/max(desc="high sale" aln=R)
*!xmlmore! purchase/min(desc="low sale" aln=R sumbgcolor?=white sumfgcolor?=teal)
CREATE SUMBGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE ' ' END
CREATE SUMFGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE ' ' END
*!xmltotal! eof  gross/max(desc="high valuation" aln=R)
*!xmlmore!  gross/min(desc="low valuation" aln=R ) purchase/max(desc="high sale" aln=R)
*!xmlmore!  purchase/min(desc="low sale" aln=R eofbgcolor?=white eoffgcolor?=teal)
CREATE EOFBGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE ' ' END
CREATE EOFFGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE ' ' END
```

The variable attribute information could be utilized in a stylesheet (e.g. an XSL file described below) to create an HTML display of various elements in different colors and or fonts , determined by values present in the data.

7.24.5.7 Special Handling of Text Fields

The XML Preprocessor uses a special syntax to identify text fields, so that REPORT can handle them in a special way that avoids potential syntax errors in the .REP file generated by the preprocessor.

In the following XML statement:

```
*!xml! fremail(desc="Submitter's email") subject msg(text desc=Feedback aln=T)
```

the grouping "msg(text desc=Feedback aln=T)" specifies that the field MSG should be output with the attributes

desc and aln set to "Feedback" and "T" respectively, as described previously, and the presence of the keyword "text" inside the attribute list identifies MSG as a field that should receive special "text field" processing.

In the generated report, the instructions for outputting field MSG and its tags end up looking like this:

```
«msg»
msg-------------------------------------1-------------------------------------
«/msg»
```

The embedded "1" in the field designator for MSG (ordinarily the posiition of the "text height" specifier for text fields) signals special processing to REPORT, avoiding the normal restrictions imposed on text fields that would prevent, for example, two text fields from being displayed in their entirety in the same DETAIL or SUMMARY paragraph.

7.24.5.8 Special Document Attributes Store Report Info

Three attributes describing the circumstances of the creation of the XML file are automatically included in the XML created by the AdmReport "-XML" command line option. This information is stored in the on-board DTD (document type definition) of the XML file as fixed default attributes of the DOCUMENT element (the root element).

The three fixed default attributes are:

```
repinfo_date:          The date when the REP or RPX was run.
repinfo_time:          The time when the REP or RPX was run.
repinfo_who:           The username that ran the REP or RPX.
```

The attributes appear in the XML file's DTD as in the following snippet:

```
<!ATTLIST DOCUMENT repinfo_date CDATA #FIXED "12-Feb-2010">
<!ATTLIST DOCUMENT repinfo_time CDATA #FIXED "13:56:22">
<!ATTLIST DOCUMENT repinfo_who CDATA #FIXED "BD">
```

7.24.5.9 The XSL Stylesheet

ADMINS Inc. provides a "generic" XSL stylesheet , admdefault4.xsl, usually kept in the same directory as the admins "exe" files, that can be used with the XML output from any report produced by the XML preprocessor. This XSL stylesheet provides a template to display the XML produced by the XML-preprocessed REPORT as an HTML table made up of one row per record and one column per field. Summary elements of various levels are displayed in contrasting colors.

This stylesheet will look for six "hardwired" element attributes.

7.24.5.9.1 DESC: column heading and summary label

If an element has the attribute "desc" it will be used as a column header and as a label in summary elements (if no desc attribute is present the element name from the XML document is used).

7.24.5.9.2 ALN: cell formatting and display

If an element has an "aln" attribute it will be used to align the element's content in the table cell (L=left, C=center, R=right, P="preformatted", T="text box"). If the aln attribute is set to "X" the element will not be displayed. If a "summary_n" element has an aln attribute set to the value "preview" it will be displayed before its child summary or record elements are displayed.

If the aln attribute is set to "link", the elements contents are rendered as a clickable hypertext link, with the target for the link specified by an another attribute for the same element that begins with the letters "href". For example:

```
*!xml!   HART(desc="Hovedart" aln=link HREFHART?="")
```

where the target of the link is to be specified in the attribute HREFHART which is blank by default and will be loaded with the contents of the field HREFHART in the REPORT virtual record.

7.24.5.9.3 CELLCLASS: color and font characteristics

If an element has a "cellclass" attribute it is used to set color and font characteristics for the table cell. The default stylesheet defines the following classes of cells:

Cellclass	Foreground	Background	Font Style
Red	Red	White	Default
Orange	Orange	White	Default
Yellow	Yellow	White	Default
Blue	Blue	White	Default
Black	Black	White	Default
Gray	Gray	White	Default
Purple	Purple	White	Default
Pink	Pink	White	Default
Teal	Teal	White	Default
White	White	Black	Default
Warn	White	Red	bold, italic

Cellclass	Foreground	Background	Font Style
Alert	Black	Yellow	bold, italic

This XML preprocessor statement specifies that field d2 should be output with the column description "money", right justified within the table cell, and displayed in orange text on a white background.

```
*!xml! d2(desc=money aln=R cellclass=orange)
```

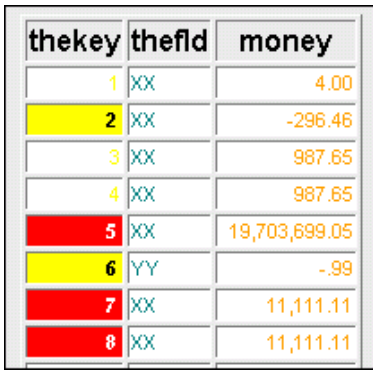
thekey	thefld	money
1	XX	4.00
2	XX	-296.46

You can use the "cellclass" attribute to make text and background color, as well as font style, **data-dependent**.

Use the XML preprocessor "?=" operator (described above) to specify different values for the cellclass attribute depending on data values, as in the following example.

This CREATE statement, in conjunction with the XML preprocessor statement that follows it, specifies that field n should be displayed as cellclass "warn" (white on red, bold, italic) if the value of field d2 is larger than 1,000, displayed as cellclass "alert" (black on yellow, bold, italic) if the value of field d2 is less than zero, and otherwise displayed as cellclass "yellow" (the default cellclass for field n).

```
create cellclass/a20 if d2 gt 1000 then 'warn' else if d2 lt 0 then
'alert' else ' ' end

*!xml! n(desc=thekey aln=R cellclass?=yellow)
```



thekey	thefld	money
1	XX	4.00
2	XX	-296.46
3	XX	987.65
4	XX	987.65
5	XX	19,703,699.05
6	YY	-.99
7	XX	11,111.11
8	XX	11,111.11

It is expected that developers will create additional cellclasses that meet their particular requirements by altering the default stylesheet, admdefault4.xsl. The section of the default stylesheet that contains the cellclass definitions looks like this:

```
<style>
  td.red    { color: red     }
  td.orange { color: orange  }
  td.yellow { color: yellow  }
  td.blue   { color: blue    }
  td.black  { color: black   }
  td.gray   { color: gray    }
  td.purple { color: purple  }
  td.pink   { color: pink    }
  td.teal   { color: teal    }
  td.white  { color: white; background-color: black  }
  td.warn   { color: white; background-color: red;
```

```
                font-weight: bolder; font-style: italic}
  td.alert    { color: black; background-color: yellow;
                font-weight: bolder; font-style: italic}
</style>
```

7.24.5.9.4 ADM_HEADING: override default document heading

If any element has an "adm_heading" attribute it is used to override the default heading ("ADMINS Web Report")

7.24.5.9.5 ADM_TITLE: override default document title

If any element has an "adm_title" attribute it is used to override the default title ("ADMINS Web Report")

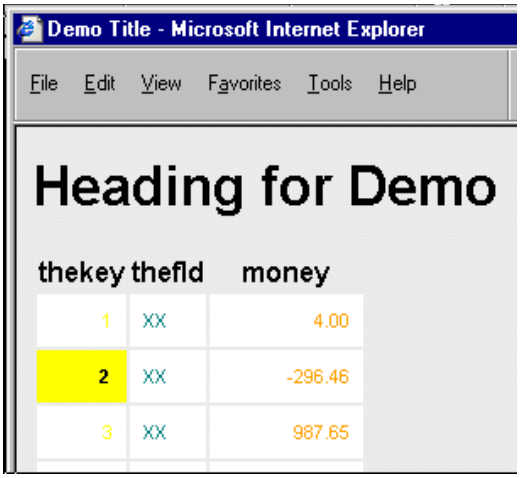
7.24.5.9.6 ADM_BORDER: suppress display of table cell borders

If any element has an "adm_border" attribute it is used to set the border width of the table cells (the default is 1).

If you set adm_border = 0 then table cell borders are suppressed.

These XML preprocessor statements specify that the document heading is "Heading for Demo", that the document title (appears in the Windows banner) is "Demo Title", and that table cell borders should be suppressed. Note that the adm_heading, adm_title, and adm_border attributes can be declared for any element - it does not matter which and they should be declared only once. Note also that when adm_border is declared cell-padding is increased.

```
*!xml! n(desc=thekey aln=R cellclass?=yellow)
*!xml! fld(desc=thefld adm_heading="Heading for Demo" adm_title="Demo Title")
*!xml! d2(desc=money aln=R cellclass=orange adm_border=0)
```



7.24.5.10 L$XSL_STYLESHEET: Specifying the stylesheet

The name of the XSL file that should be used to display the XML file produced by the XML Preprocessor is written into the XML file when it is created. Specify the XSL file you want to be used by assigning its path specification to the logical name L$XSL_STYLESHEET, e.g. to indicate that the generic stylesheet supplied by ADMINS is to be used you would make a logical name assignment similar to:

```
Admlcr l_xsl_stylesheet c:\progra~1\admins\bin\admdefault4.xsl
```

To use a different stylesheet, just load its path into the L$XSL_STYLESHEET logical:

```
Admlcr l_xsl_stylesheet stylecolor.xsl
```

As the preprocessor-generated .REP uses an ADMINS logical parameter <L$XSL_STYLESHEET>, if this logical name is not assigned REPORT will prompt for it. If no path information is provided with the file name, as in the above example, the XML interpreter will expect the XSL file to be in the same location as the XML file.

7.24.5.10.1 Identifying the stylesheet inside the .REP file

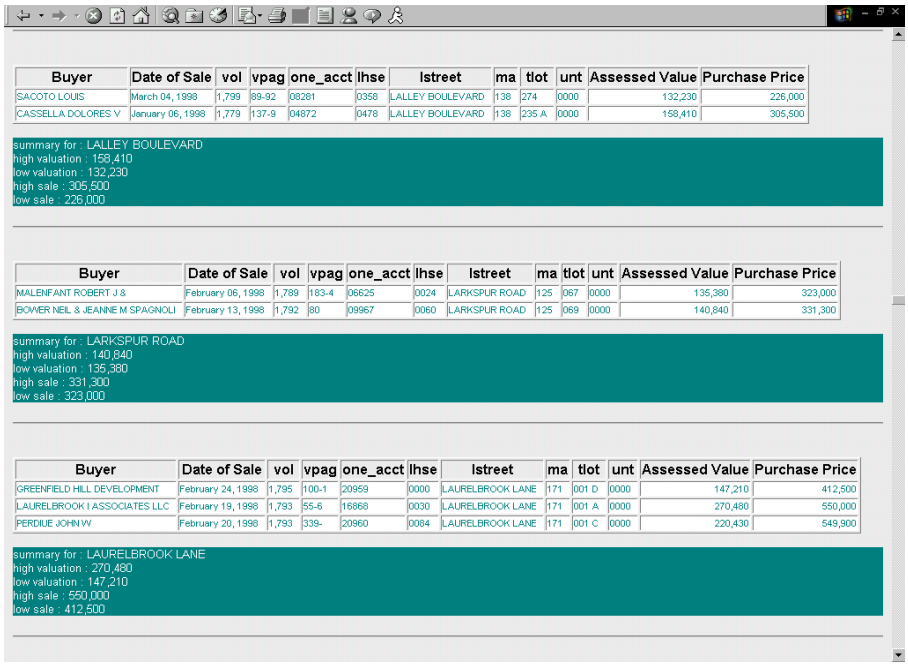
Alternatively, the XSL file can be named inside the .REP, using the "*!xmlStyleSheet!" preprocessor statement, as follows:

```
*!xmlStyleSheet! c:\bills\admdefault4lw.xsl
```

If the *!xmlStyleSheet! statement is present the L$XSL_STYLESHEET logical name is not checked and REPORT will not prompt if it is not assigned. "Hardcoding" the stylesheet specification in this manner ensures that no prompting for the unsupplied logical parameter is attempted (a valuable feature for automatic scripts that are used for instance in the webserver environment).

7.24.5.10.2 Example: using the "default" stylesheet

Here's an example of what an XML file looks like when displayed using the ADMINS' default XSL stylesheet (of course you can write your own XSL to display the generated XML in any way you like). For a good place to start learning about XSL, try http://www.w3schools.com/xsl/.



Buyer	Date of Sale	vol	vpag	one_acct	lhse	lstreet	ma	tlot	unt	Assessed Value	Purchase Price
SACOTO LOUIS	March 04, 1998	1,799	89-92	08281	0358	LALLEY BOULEVARD	138	274	0000	132,230	226,000
CASSELLA DOLORES V	January 06, 1998	1,779	137-9	04872	0478	LALLEY BOULEVARD	138	235 A	0000	158,410	305,500

summary for : LALLEY BOULEVARD
high valuation : 158,410
low valuation : 132,230
high sale : 305,500
low sale : 226,000

Buyer	Date of Sale	vol	vpag	one_acct	lhse	lstreet	ma	tlot	unt	Assessed Value	Purchase Price
MALENFANT ROBERT J &	February 06, 1998	1,789	183-4	06625	0024	LARKSPUR ROAD	125	067	0000	135,380	323,000
BOWER NEIL & JEANNE M SPAGNOLI	February 13, 1998	1,792	80	09967	0060	LARKSPUR ROAD	125	069	0000	140,840	331,300

summary for : LARKSPUR ROAD
high valuation : 140,840
low valuation : 135,380
high sale : 331,300
low sale : 323,000

Buyer	Date of Sale	vol	vpag	one_acct	lhse	lstreet	ma	tlot	unt	Assessed Value	Purchase Price
GREENFIELD HILL DEVELOPMENT	February 24, 1998	1,795	100-1	20959	0000	LAURELBROOK LANE	171	001 D	0000	147,210	412,500
LAURELBROOK I ASSOCIATES LLC	February 19, 1998	1,793	55-6	16868	0030	LAURELBROOK LANE	171	001 A	0000	270,480	550,000
PERDUE JOHN W	February 20, 1998	1,793	339-	20960	0084	LAURELBROOK LANE	171	001 C	0000	220,430	549,900

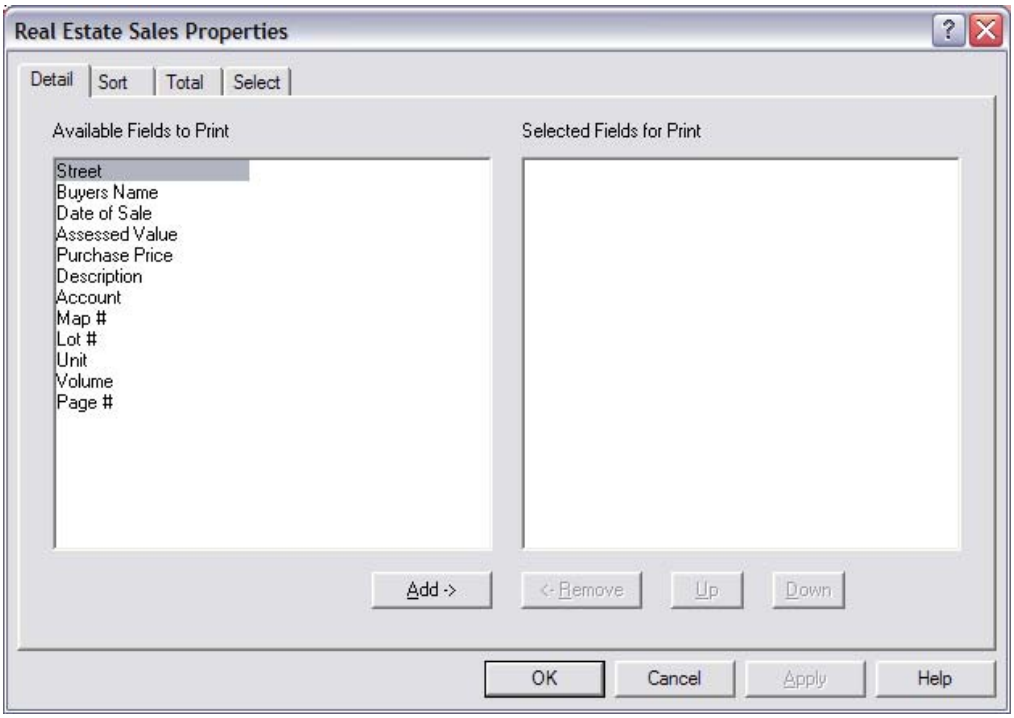
summary for : LAURELBROOK LANE
high valuation : 270,480
low valuation : 147,210
high sale : 550,000
low sale : 412,500

7.25 AdmRG: ADMINS Report Generator

The ***ADMINS Report Generator*** (**AdmRG**) allows the developer to use enhanced ADMINS REPORT syntax to build a "flexible reporting environment" for users. Browser-oriented (XML), spreadsheet-oriented (CSV), or data-interchange (CSV or XML) outputs can be created by the user according to their changing requirements by selecting open-ended combinations of the report building blocks put in place by the developer.

AdmRG provides the user with a list of fields available for display at the detail level, fields available to use for sorting and selecting data, and possible break levels with summary fields available for display. If the specified format is *XML* a browser is launched after the report is generated to view, and possibly print, the report.

When an AdmRG report is requested the user is presented with the ***ADMINS Report Generator Property Sheet***:



It consists of four property sheets:

Detail to select fields to be included in the *DETAIL* section of the report.

Sort to specify the order in which the information should be presented (e.g. by Street Name, by Account Number, by Last Name, etc.).

Total to select possible total breaks, and which fields should be presented at each break level (available breaks depends on the selected sort order).

Select to enter logic for which records should be selected for printing.

Note that clicking either of the buttons at the bottom will terminate the dialog. Click on OK to generate the selected report. Click on Cancel to "back out" of the AdmRG dialog (exits without saving any of your selections or running the report).

7.25.1 Calling AdmRG

AdmRG can be called from the command line:

```
admrg
```

or

```
admarg  myfiles:myrpx.rpx
```

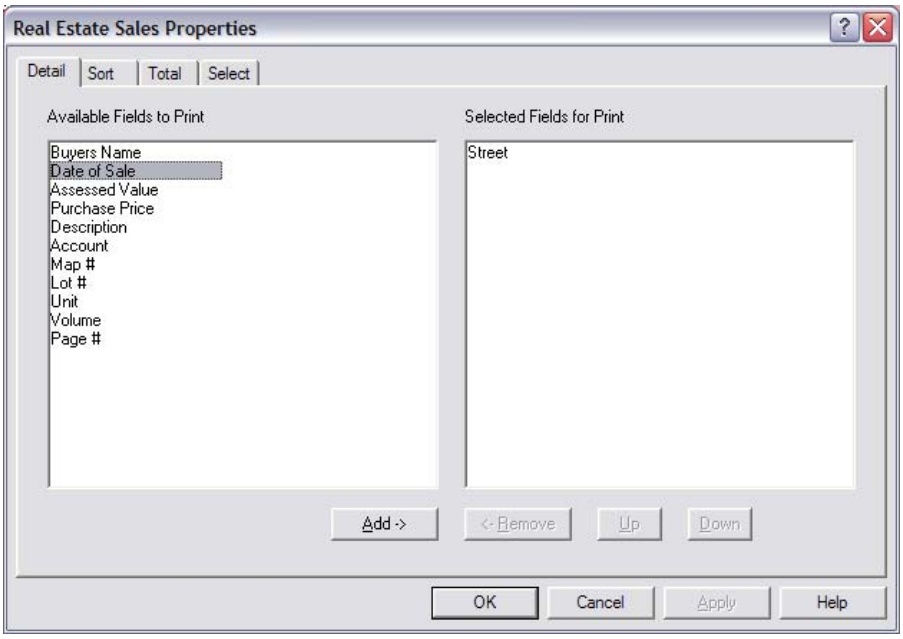
When called without any argument AdmRG will present file selection a dialog box in which the user may select an .RPX (see Section 7.25.6 "The AdmRG Macro Language.")

AdmRG may also be called from AdmTrans using AppMenu (see Appendix M: "The AppMenu SubSystem").

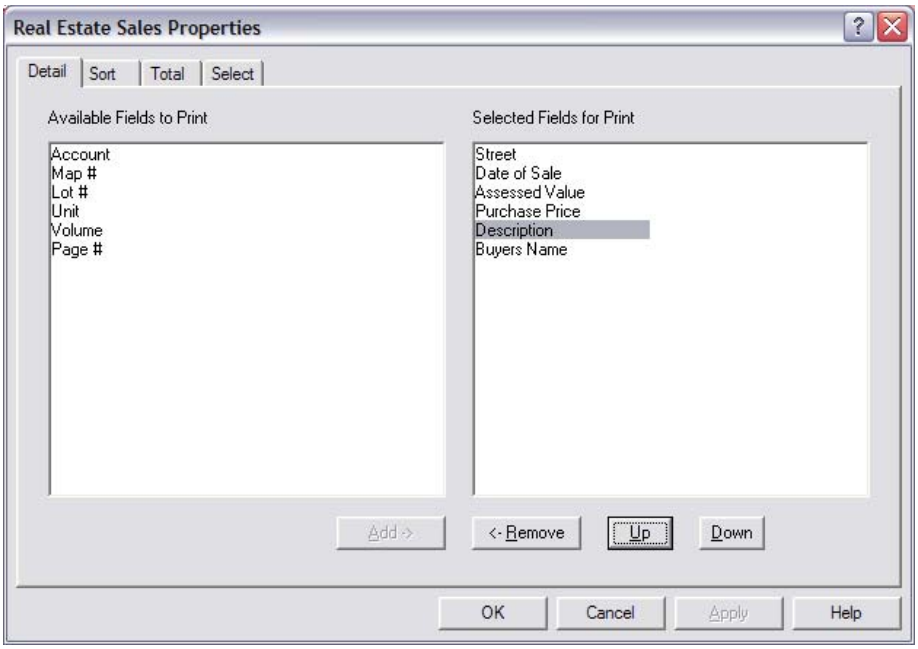
7.25.2 DETAIL Property Sheet

The first Property Sheet presented lists the detail-level fields available (corresponding to the *DETAIL* section in a standard ADMINS REPORT instruction file).

When a field in the ***Available Fields to Print*** listbox is selected the Add -> button is activated, allowing the user to move the field from the *Available* to the *Selected* listbox.



After moving fields over to the *Selected* listbox you may also select one of those fields:

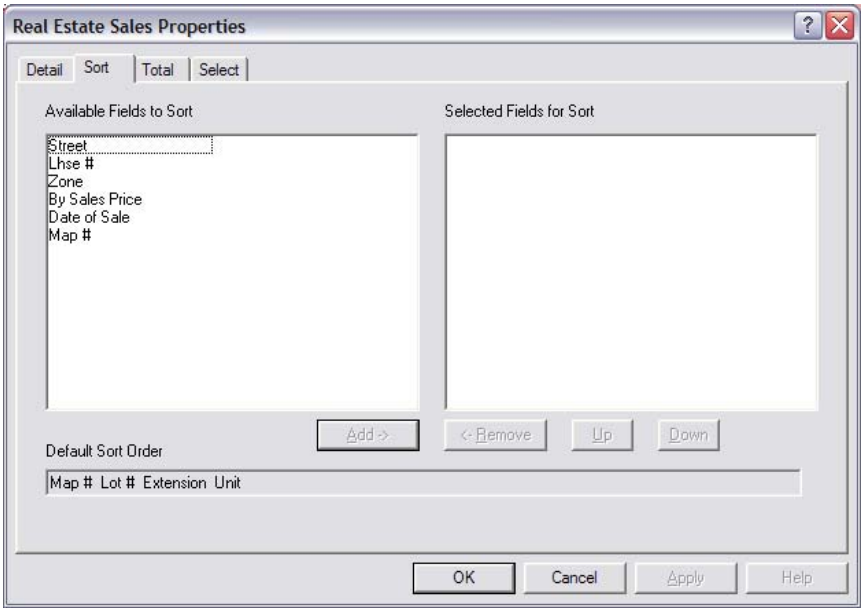


The <- Remove button is now activated, allowing you to remove the field from the *Selected* to the *Available* listbox.

The fields will be listed in the order they appear in the *Selected* listbox. Observe that the Up and Down buttons are activated, making it possible to move the selected item up or down in the list of selected fields, thus altering the order in which they will appear on the printed report.

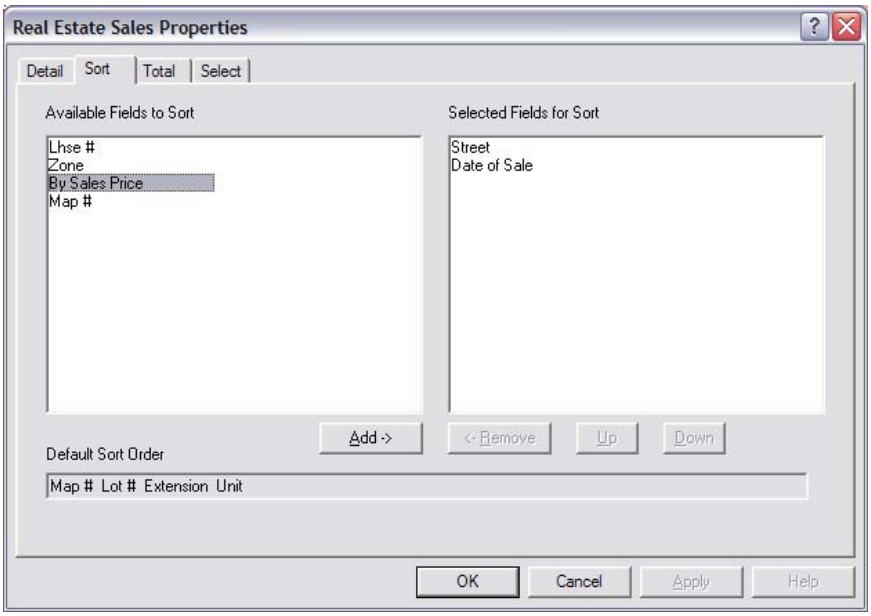
7.25.3 SORT Property Sheet.

The ***Sort*** *Property sheet* presents a list of fields that the user may select from to alter the order in which the data will be presented.



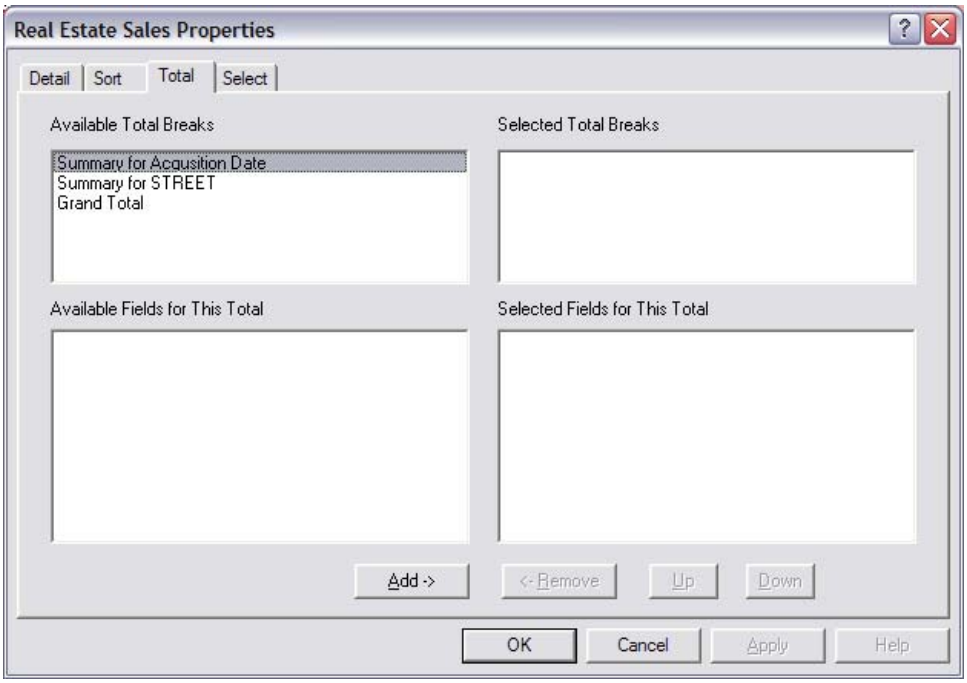
The *Default Sort Order* control shows the order records will appear in if no sort fields are selected. This is usually the key order of the primary file of the data view being presented.

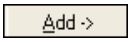
The *Sort Order* selected determines which *TOTAL breaks* will be available for selection in the ***Total*** *Property Sheet*. It is therefore important that any sort order fields be selected before activating the ***Total*** *Property Sheet*.

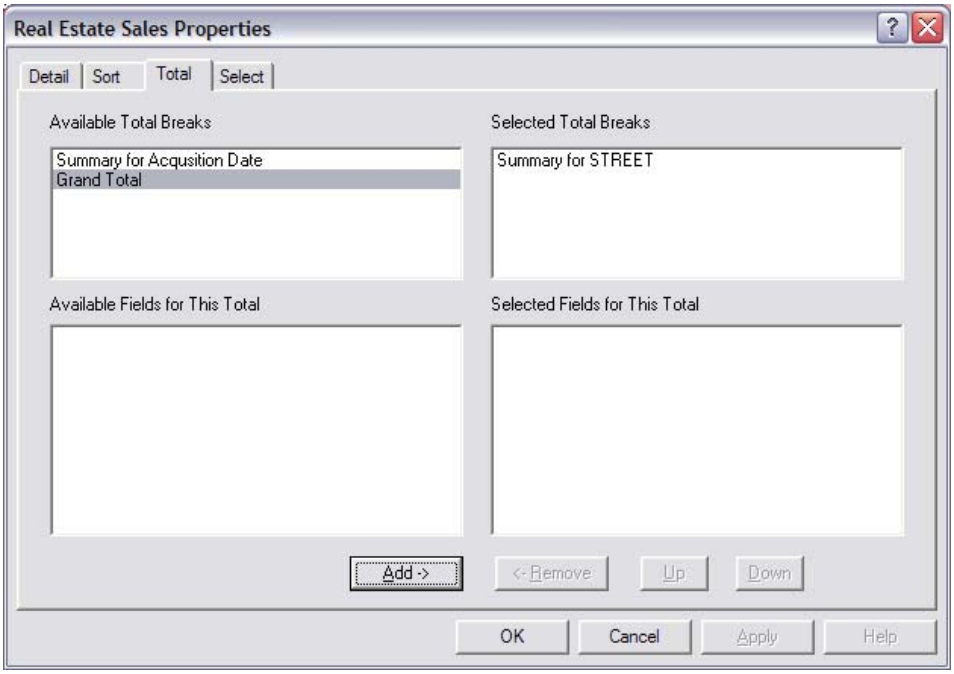


7.25.4 TOTAL Property Sheet.

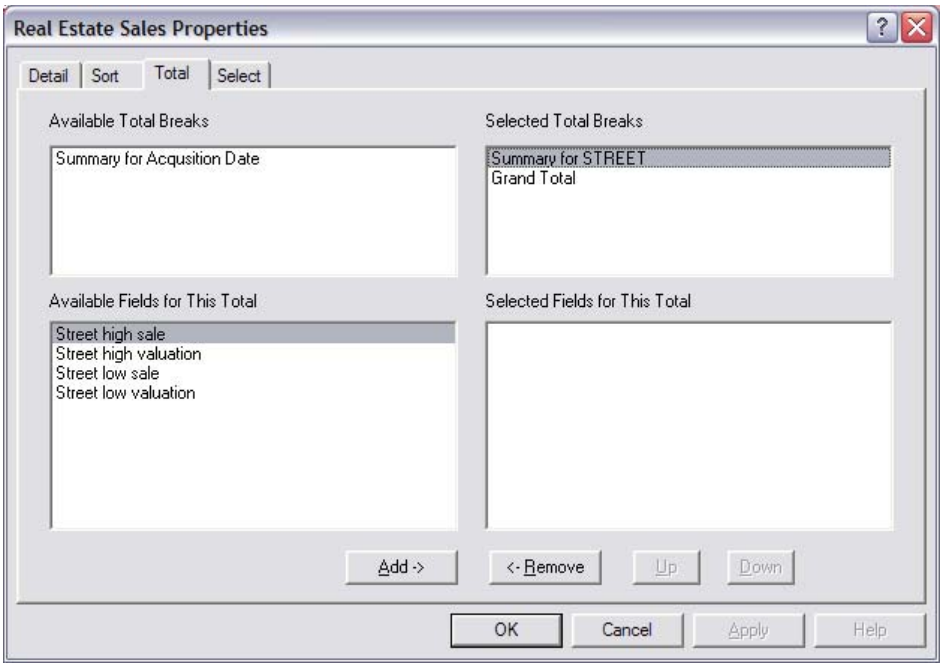
Depending on the sort order that is in effect (either specified in the *SORT* property sheet, or by the *default sort order* of the virtual record) one or more possible total (or summary) breaks may be presented.



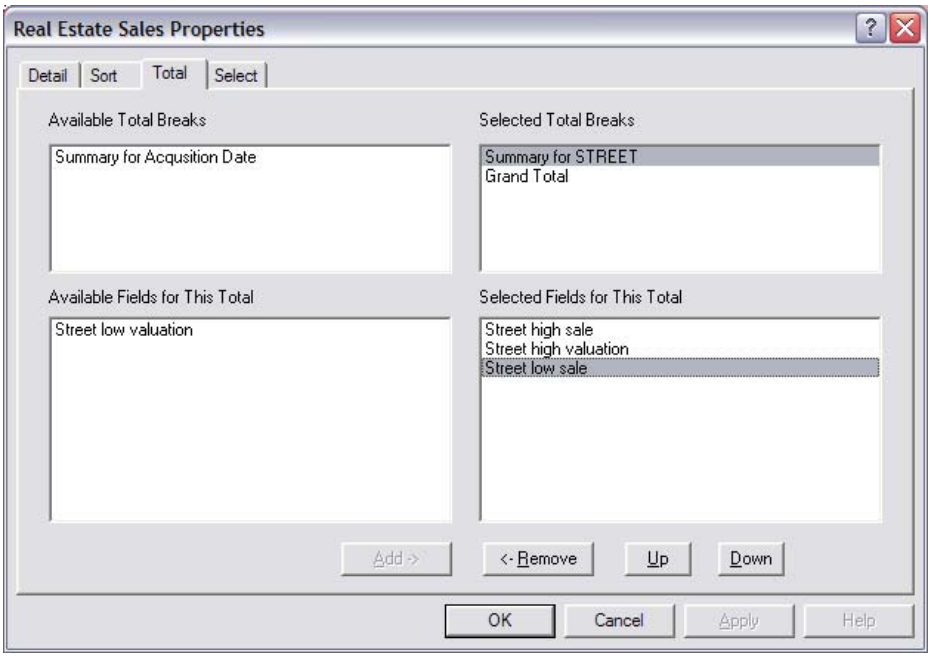
By selecting one of them the  button becomes available, allowing you to move the break level from the *available* to the *selected* listbox.



Once you have selected the total break levels you want, **select** one of them and the *Available Fields* for that total break becomes visible:



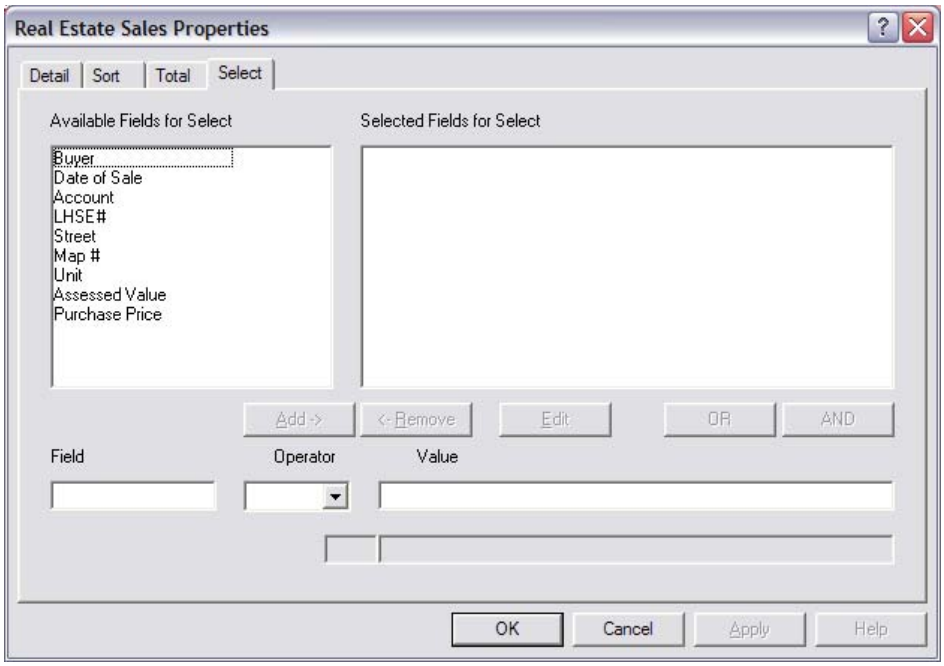
You may now select and add fields to the *Selected Fields for This Total*.



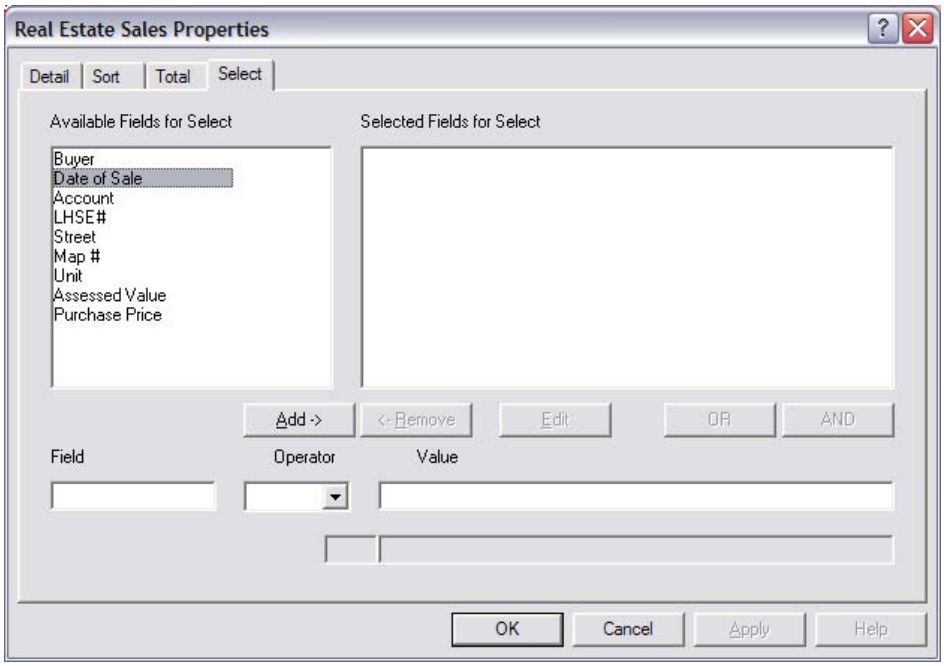
If you select a field in the *Selected Fields for This Total* listbox, the <- Remove, Up and Down buttons become available to remove or move the field.

7.25.5 SELECT Property Sheet.

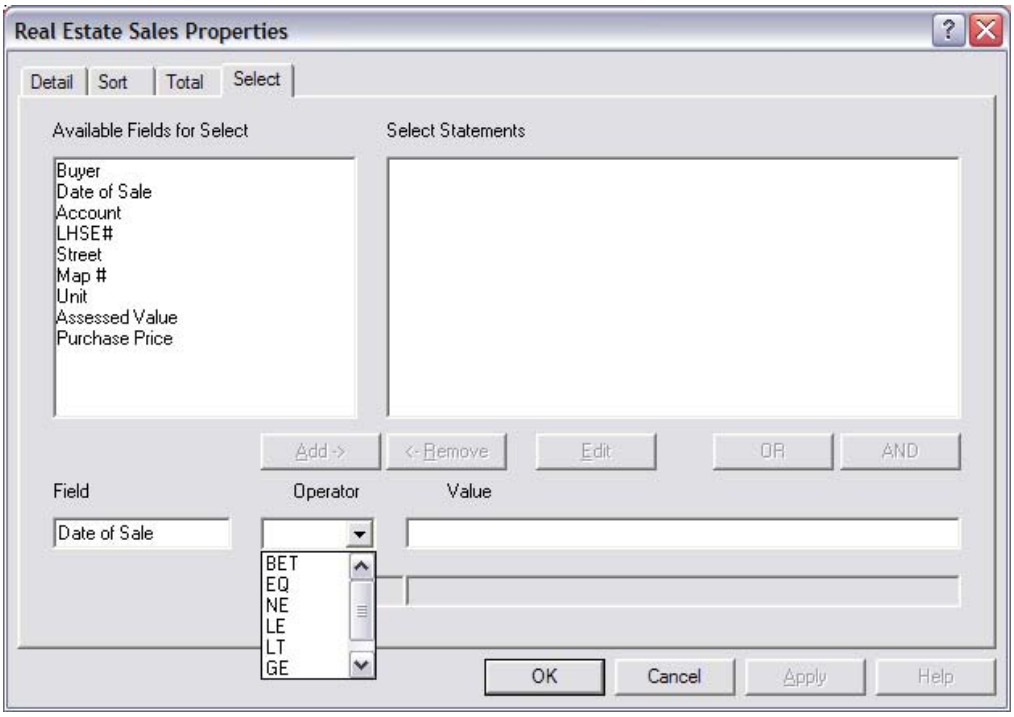
The ***Select*** *Property Sheet* is used to enter selection criteria for which records should be included in this report. The initial window shows which fields are available for selection logic;



Select a field in the *Available* listbox, and the Add -> button becomes active:



Press the Add -> button to copy the selected field to the *Field* edit control. When you do this the *Operator* combobox will be activated:



You now have to select one of the Boolean operators to apply to the selected field. Available operators are:

BET The value in the virtual record has to be between two entered values, the entered values included.

EQ The value in the virtual record has to exactly match the entered value.

NE The value in the virtual record has to different from the entered value.

LE The value in the virtual record has to be less than or equal to the entered value.

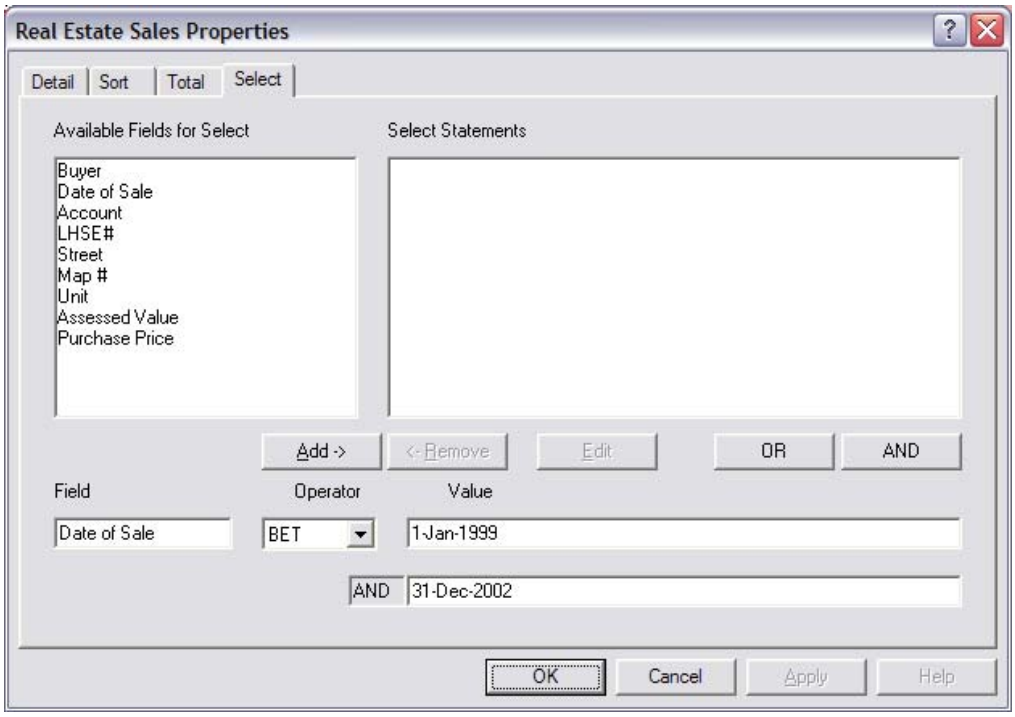
LT The value in the virtual record has to be less than the entered value.

GE The value in the virtual record has to be greater than or equal to the entered value.

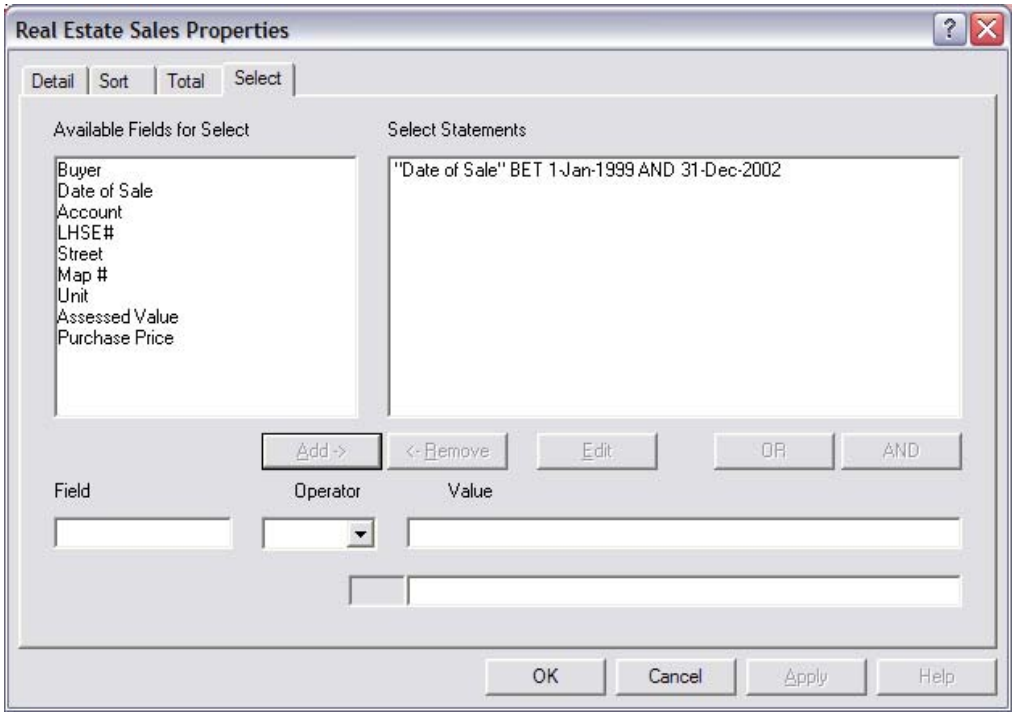
GT The value in the virtual record has to be greater than the entered value.

INCL The field value must include the specified value

Once an operator is selected the cursor moves to the value field, allowing the user to enter a value. Terminate the entry by hitting the TAB key. If the operator was *BET* the next value field will be activated, and the operator *AND* will be inserted between the two values.



Once the necessary values are entered (terminated by the *TAB* key) buttons are activated to move the selection statement to *Selected Statement* listbox, and the edit controls are cleared.



7.25.6 The AdmRG Macro Language.

The developer puts "report building blocks" in place: the "virtual record", what fields are available; what selection criteria are available; what choice of sort orders are available, what levels of aggregation are available, and what values can be presented at those aggregate levels. These building blocks are put together in an AdmRG specification named with the extension "***.RPX"***, a simple text file which defines the data view and specifies the choices for this AdmRG.

AdmRG uses standard REPORT syntax (i.e. standard REPORT FILE, LINK, TABLE and CREATE/COMPUTE statements) to specify the virtual record. As a general rule, standard REPORT syntax is passed on through the AdmRG preprocessor engine without modification.

Users build their own reports by combining the building blocks made available to them in whatever way it takes to solve their problem, meet their requirement, or answer their question. User choose the fields they want (Detail or Total) from the records they want (Select) presented in the order they want (Sort) aggregated the way they want.(Total).

AdmRG macros are used to specify the choices that will be available to the user. The general format of an *AdmRG macro* is:

```
*!keyword! FLDNAME …
```

Largely the macros follow the same rules as used for ADMINS XML reports.

The general way of accessing a field from the virtual record is:

```
FLDNAME
or
FLDNAME(attributes)
```

where *attributes* are any keywords recognized by the style sheet in use, normally used to modify the way a field value is displayed. The only attribute recognized and used by *AdmRG* is ***desc*** which is used to provide a user friendly name for the field.

An example would be:

```
ACCNT
or
ACCNT(desc="Customer account #")
```

In the first case the field name "ACCNT" would show up as the heading (or label) of the field, while in the second case you would get the more appropriate "Customer account #".

The ***desc*** attribute has 3 possible formats:

desc="text" where ***text*** becomes the descriptive text used for the field.

desc?="text" where ***text*** is the default descriptive text being used, but if a field named **DESC** exists and have a value, that value will be used instead.

desc ***desc*** by itself means get the ***LABEL*** value from the Data Dictionary and use it as its descriptive text.

Some times it may be preferable to present the user with a descriptive name for a field other than what will be passed on in the XML output using the *desc*= attribute. This can be obtained by using the special attribute ***argdesc=***, e.g.

```
ACCNT(argdesc="Customer Account" desc="Account #")
```

In this case the user will see the name *Customer Account* in the AdmRG property sheets, while it will show up as *Account #* in the XML file. You may also use:

```
ACCNT(argdesc="Customer Account" desc=" ")
```

in which case no descriptive text will show up in the XML file.

The *argdesc*= argument is removed before the attribute list is passed on in the .REP file. This also means that if you use e.g.

```
ACCNT(argdesc="Customer Account")
```

i.e. *argdesc*= is the only attribute, no attributes will be passed on to the report.

The use of *argdesc*= might be especially useful in an **!argtotal!* statement, e.g.

```
*!argTotal! lstreet(argdesc="Summary for Street" desc="Summary for")
```

In this case the user will see *"Summary for Street"* in the *Available Total Breaks* listbox, while it will show up in the XML report as e.g. *Summary for MAIN STREET*, i.e. the literal text *Summary for* followed by the actual name of the street causing this total break.

7.25.7 Specifying the Virtual Record

RVirtual record specification in the AdmRG instruction file (.RPX) follows standard REPORT rules. Any FILE, LINK, TABLE and CREATE (COMPUTE) statement will be passed on as specified (including CREATE (COMPUTE) statements in TOTAL sections).[62]

7.25.8 The AdmRG Macros

AdmRG Macros are used to specify which fields are available for the user to choose from in four different areas of the report:

The ***DETAIL*** section, specifying which fields are to be reported for each selected virtual record.

The ***SORT*** statements, specifying in which order the information will be presented, and also determining which summary breaks will be available (you may only have a summary break at a field contained in the sort order of the report, either implicit by the key structure of the file, or explicitly specified in a *SORT* statement).

The ***TOTAL*** section, specifying which cumulative values will be presented at a given summary level.

---

62. The FILE statement in an RPX can reference an RMO, i.e. all local fields in the RMO can be used as fields in the RPX just as in a regular ".REP".

    If FILE references an RMO you must include EXECUTE statements the same way as they are used in REPORT (the .REP created by ARG will be executed by REPORT).

The ***SELECT*** statements, specifying which virtual records should be selected for presentation.

An *AdmRG* report may be used to produce an ***XML*** or a ***CSV*** output. All keywords used by *AdmRG* will therefore start with the letters ***xml*** or ***csv*** to indicate which output type they are intended for. But in many cases a macro can be used for both types of output, and therefore, if the keyword starts with ***arg*** the macro will be included for both types.

7.25.8.1 DETAIL Section

To specify fields available for printing for each selected virtual record the ****!arg!*** keyword is used (use ****!xml!*** if it is intended for XML only, or ****!csv!*** if only intended for CSV). E.g.:

```
*!arg! lstreet(desc=Street) name(desc=Buyer)
*!arg! acqdate(desc="Date of Sale")
*!arg! gross(desc="Assessed Value" aln=R)
```

7.25.8.2 SORT Section

The *SORT* section uses two keywords:

```
*!argsortdefault! fieldname …
```

to specify what the default sort order is. This statement is informational only, and is only used to provide *AdmRG* with information about which summary breaks available to present to the user. The field names specified in this statement must be the key fields in the primary file in the report.

```
*!argsort! fieldname …
```

This specifies the fields that are available for the user to sort on.

If a field available for *SORT* is also available in the *DETAIL* section it is not necessary to provide any *(desc="Description")* attribute for the field, as it will be picked up from the *DETAIL* section. You may use a *(desc="Description"*) attribute if you want to override the description from the *DETAIL*.

7.25.8.3 TOTAL Section

The developer should provide a *TOTAL* section for every possible sort break made available to the user in the SORT Section. The keywords used to specify which total breaks are available and which fields are available at each total level are ****!argtotal!*** (or ****!xmltotal***) to specify the break field and fields available for display , and ****!argmore!*** (or ****!xmlmore!***) to specify more available fields at this total break than will fit on the **!argtotal!* line. E.g.:

```
*!argtotal! lstreet(desc="Summary for Street" aln=C)
*!argmore! gross/max(desc="Street high valuation" aln=R)
*!argmore! gross/min(desc="Street low valuation" aln=R)
```

7.25.8.4 SELECT Section

This is where the user will specify which virtual records to include in the final report. The syntax is:

```
*!argselect! fieldname …
```

Every field name listed will be available for the user to specify a "filter" for selecting records, by selecting a field name (by its user-friendly description if available) and the "filter" for it, specified using a boolean operator and a test value, e.g.:

```
"Last Name" EQ 'Washington'
```

7.25.9 Default DESC Attributes.

The ***desc*** attribute should always be present for a field in the *DETAIL* section (unless you want to use the field name (in uppercase) as the field descriptor). If no **desc** (or **argdesc**) attribute is available for a field in any of the other sections it will attempt to get its descriptor from the *DETAIL* section. Thus, for this other sections it is only necessary to provide ***desc*** attributes if you want the descriptive text to be different from what you provide for the *DETAIL* section.

Some special rules apply for the *TOTAL* section. If a field available in a *TOTAL* section has an aggregation operator attached to it the ***desc*** text obtained from the *DETAIL* section will be modified depending on the aggregation operator in effect. E.g. if the *DETAIL* section contains:

```
salval(desc="Sales Value")
```

and a *TOTAL* section contains e.g. *SALVAL/MIN* or *SALVAL/AVG* without any ***desc*** attributes specified, the ***desc*** values being used are:

salval/E	"Non-Null Sales Value"
salval/AVG	"Avg. Sales Value"
salval/MAX	"Max Sales Value"
salval/MIN	"Min Sales Value"
salval/FI	"First Sales Value"
salval/2	"Second Sales Value"
salval/3	"Third Sales Value"
salval/4	"Fourth Sales Value"
salval/LA	"Last Sales Value"

7.25.10 Special Keywords

In addition to the keywords used to specify available options in the various REPORT sections, a number of options are available to modify the behavior of the *AdmRG* generator. These are (the keywords are case blind, a mixture of lower and upper case letters are used for readability only):

The first *AdmRG* keyword present should be one of the following to specify which report types are present, or possible to generate from this AdmRG specification

Keyword	Description
***!argXml!** **or** ***!argCsv**	This AdmRG specification can only produce XML or only CSV output
***!argXmlCsv!** **or** ***!argCsvXml!**	This AdmRG specification can produce both XML and CSV output. If no report type is specified when AdmRG is activated, the user is prompted for which type to generate.
***!argTitle!**	Title Name This will show up in the Caption and the Heading of the produced XML report.
***!argRepName!**	Output Report Name This may be used to name the .REP generated by AdmRG. By default the generated .REP is named the same as the .RPX macro file being used (including the directory path), with a .REP extension instead of the .RPX. The report name may contain the ***$keyword$*** variables also supported by Adm2Perl. E.g.: ***!argRepName! MYDIR:Sales_$USERNAME$.rep** will create the report in MYDIR and insert your Username as part of the report name. The special keyword **ARG** is supported in AdmRG to be able to insert the type of report being produced as part of the file name,e.g.: ***!argRepName! Sales_$USERNAME$_ARG.rep** will replace the ARG with either XML or CSV depending on which type of report being requested. As selected values are loaded from the previous run of an AdmRG report it will be important to be able to distinguish between the two possible types of reports.
***!argOutName!**	Pathname of output file This may be used to name the output file from report. This will have the same effect as running the report as AdmReport /xml=filename. ***!argOutName MYDIR:CurrentSales_$USERNAME$** It is not necessary to provide any file type for the output file name, as AdmRG will add the file extension .xml or .csv depending on what type of output being produced.

Keyword	Description
***!argOutVersions!**	Specify number of versions of output file are to be kept, e.g.: ***!argoutversions!10** specifies that the output of the last 10 runs of this RPX will be kept.
***!xmlstylesheet!**	Pathname of style sheet This statement may be used to specify that the resulting XML report should use the specified style sheet rather than the one identified by the L_XSL_STYLESHEET logical name.

7.25.11 Sample AdmRG Instruction File.

```
*!argxmlcsv!
*!argtitle! Real Estate Sales
*!argRepName! ADM_ARG:Demo1_$USERNAME$_$ARG$.rep
*!argOutName! ADM_ARG:Demo1_$USERNAME$
*~
REPORT RESALE
*~*************************************************************************
*~ Regular FILE, LINK and COmpute statements are used to create the
*~ virtual record for the report.
*~-------------------------------------------------------------------------
FILE ADM_ARG:RE.MAS-R
LINK NAME FROM ADM_ARG:REACCOUNT.MAS KEY IS 1ACCT
CO BL/A1
CO TLOT/A6 NCAT(TLOT,LOT,BL,EXT)
CO COLOR/A10 IF PURCHASE GT 500000 THEN 'YELLOW' ELSE ' ' END
CO FONT/A2 IF ACQDATE GT 1Apr1998 THEN '16' ELSE ' ' END
*
*************************************************************************
*  Possible selections
*~ Listing of fields the user can use for selection
*-------------------------------------------------------------------------
*!argselect! name acqdate 1acct(desc="Account") lhse# lstreet map lot unt
*!argselect! gross purchase
*
*************************************************************************
*  Possible SORT fields:
*~ The desc attributes are only present for display purposes,
*~ and will not be present in the resulting sort statement.  If the
*~ desc attributes are the same as in the DETAIL section it is not
*~ necessary to repeat them here.
*~
*~ The *!argsortdefault! lists the defualt sort order provided by the
*~ main file key structure.
*~
*~ The *!argsort! statements lists all the fields users are allowed to
*~ pick for sorting the outputt.
*-------------------------------------------------------------------------
*!argsortdefault! map lot ext(desc=Extension) unt
```

```
*!argsort! lstreet lhse#(desc="Lhse #") zone(desc=Zone)
*!argsort! purchase(desc="By Sales Price") acqdate map
*
*************************************************************************
* Detail fields:
*~
*~ Using *!arg! instead of *!xml! or *!csv! allow these statements to
*~ be used both for XML and CSV reports.
*-----------------------------------------------------------------------
*!arg! lstreet(desc=Street)
*!arg! name(argdesc="Buyers Name" desc="Buyer") acqdate(desc="Date of Sale")
*!arg! gross(desc="Assessed Value" aln="R")
*!arg! purchase(desc="Purchase Price" aln="R" COLOR?=red font?=12)
*!arg! desc(desc=Description) 1acct(desc=Account)
*!arg! map(desc="Map #") lot(desc="Lot #") unt(desc=Unit)
*!arg! vol(desc=Volume) vpag(desc="Page #")
*************************************************************************
*  Possible TOTAL statements.
*~ The selected SORT fields will determine which ones becomes available
*~ to select from.
*-----------------------------------------------------------------------
*************************************************************************
* TOTAL LSTREET Only available if LSTREET is selected as a sort field
*-----------------------------------------------------------------------
*!argtotal! lstreet(argdesc="Summary for STREET" desc="Summary for" aln=C)
*!argmore! gross/max(desc="Street high valuation" aln=R)
*!argmore! gross/min(desc="Street low valuation" aln=R )
*!argmore! purchase/max(desc="Street high sale" aln=R)
*!argmore! purchase/min(desc="Street low sale" aln=R s1bgcolor?=white
s1fgcolor?=teal)
CO S1BGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE ' ' END
CO S1FGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE ' ' END
*************************************************************************
* TOTAL ACQDATE Only available if ACQDATE is selected as a sort field
*-----------------------------------------------------------------------
*!argtotal! acqdate(desc="Summary for Acqusition Date" aln=C)
*!argmore! gross/max(desc="ACQDATE high valuation" aln=R)
*!argmore! gross/min(desc="low valuation" aln=R ) purchase/max(desc="high
sale" aln=R)
*!argmore! purchase/min(desc="low sale" aln=R s2bgcolor?=white
s2fgcolor?=teal)
CO S2FGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE ' ' ENDCO
S2BGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE ' ' END
*************************************************************************
* TOTAL MAP Only available if MAP is selected as sort a field
*-----------------------------------------------------------------------
*!argtotal! map(desc="Summary for MAP" aln=C) gross/max(desc="MAP high
valuation" aln=R)
*!argmore! gross/min(desc="low valuation" aln=R ) purchase/max(desc="high
sale" aln=R)
*!argmore! purchase/min(desc="low sale" aln=R s3bgcolor?=white
s3fgcolor?=teal)
CO S3BGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE ' ' END
CO S3FGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE ' ' END
*************************************************************************
* TOTAL ZONE Only available id ZONE is selected as sort a field
*-----------------------------------------------------------------------
*!argtotal! zone(desc="Summary for ZONE" aln=C)
*!argtotal! gross/max(desc="ZONE high valuation" aln=R)
```

```
*!argmore! gross/min(desc="low valuation" aln=R ) purchase/max(desc="high
sale" aln=R)
*!argmore! purchase/min(desc="low sale" aln=R s4bgcolor?=white
s4fgcolor?=teal)
CO S4BGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE ' ' END
CO S4FGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE ' ' END
*************************************************************************
* TOTAL EOF is always available:
*------------------------------------------------------------------------
*!argtotal! eof(desc="Grand Total")  gross/max(desc="EOF high valuation"
aln=R)
*!argmore!  gross/min(desc="low valuation" aln=R )
*!argmore! purchase/max(desc="high sale" aln=R)
*!argmore!  purchase/min(desc="low sale" aln=R eofbgcolor?=white
eoffgcolor?=teal)
CO EOFBGCOLOR/A10 IF purchase/min gt 1000000 then 'Yellow' ELSE ' ' END
CO EOFFGCOLOR/A10 IF purchase/min gt 1000000 then 'Red' ELSE ' ' END
```

Chapter 8:Expressions

There are a number of situations in ADMINS where expressions are used. These include:

1. Record maintenance procedures (RMO's) used with TRANS, MAINT, MOVE, PROD, and REPORT
2. SELECT, CREATE, and RECODE statements in a report instruction file (REP)
3. Virtual and Message fields and the Check statements in a screen instruction file (TRS)
4. SELECT statements in a file definition (DEF)

The syntax and operators used in all these situations are essentially the same, with the exception that there are certain constructions (e.g. GOTO, labels, etc.) usable only in record maintenance procedures. These exceptions are reviewed in Chapter 9: "CMP: The Record Maintenance Compiler" on the Record Maintenance Compiler.

8.1 Constants

Constants may appear in any expression wherever a field name would be valid. The general convention for specifying a constant and its type is 'VALUE/TYPE' where VALUE is a string of data type TYPE and enclosed in apostrophes. TYPE can be any data type (Ln, Dn, Fn, I, DA, DT, TM, An or Xpic) as described in Section 2.4.2 "Field Data Types", or a reference to a Data Dictionary element may be substituted for the field type specification as described in Section 1.3.5 "Referencing Data Dictionary Elements".

The type of the constant is determined by the following rules.

1. If the constant contains a "/TYPE" designation then the type is taken as designated. For example:

```
PERCENT = ( AMOUNT * '100/D' ) / TOTAL
```

2. If the constant contains a decimal point and no "/TYPE" designation, the constant is taken as a Dn where n is determined by the number of places to the right of the decimal point. That is, "100." is D but "100.1" is D1 and "100.10" is D2.
3. If the constant does not contain a "/TYPE" designation or a decimal point, the type is **assumed** to be the same as the type of the field (or constant) that appeared to the immediate left of the constant whose type is being determined.

Note the use of the apostrophes in the constant '100/D'. The apostrophes are necessary to determine whether a value is a constant or a field name. In the expression:

```
NAME = JOHN
```

JOHN is assumed to be a field name and is only treated as a constant if there is no field named JOHN. However, in the expression:

```
NAME = 'JOHN'
```

'JOHN' is always a constant. Numeric values such as 100 or 123.45, which are not valid field names because they begin and end with a digit, need not be enclosed in apostrophes.

When ADMINS analyzes a string in an instruction file to determine whether it is a field name, constant or operation, the following rules are used:

1. If the string is an operator (e.g. "+", "OR", "GT"), or punctuation (e.g. left parenthesis), or an operation name (e.g. GOTO), then ADMINS treats it as such.
2. If the string begins and ends with a digit, then the string is treated as a constant. Also, if the string contains a slash it is treated as a constant of the type designated after the slash.
3. Otherwise, the string is evaluated as an actual field name or as a local field name. If in fact the string is not the same as a field name, then it is treated as a constant. Remember, by the time ADMINS is analyzing expressions both the file to be read and all local fields are fully known.

Note, a "string" is defined as:

1. A sequence of consecutive non-blank characters or
2. those characters, including blanks, enclosed in apostrophes.

8.2 Arithmetic Operators

The arithmetic operators[1] used in ADMINS are:

Operator	Function
+	add
-	subtract
*	multiply
/	divide and round
//	divide and truncate
%	modulus

1. The "modulus" expression "X % Y" produces the remainder when X is divided by Y, where X and Y are any ADMINS numeric field types. If necessary, the result is rounded.

These operators are for use only with numeric field types, i.e., integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) type fields. All the fields in an arithmetic expression must be of the same type and may never be mixed, although L, D or F fields may be defined with a different number of decimal places. However, data types can be converted one to another using the NCAT subroutine described in Appendix H.3.2 "NCAT - Converting Between Field Types".

8.2.1 Decimal Operations

Internal intermediate results preserve the degree of precision of the most precise field or constant in the expression. (That is, the field with the greatest number of decimal places.)

For example, if FLD3 (type D3) equals 3.333 and FLD2 (type D2) equals 1.16; then if FLD1 is type D1 the addition FLD1 = FLD3 + FLD2 produces the result FLD1 = 4.5. The internal maintained as type D3 field would be 4.493, and it is rounded to 4.5 because the result field is a D1.

If FLD2 equals 2.22 and FLD1 equals 1.1, then in the expression FLD1 = FLD2 * FLD1, the internal result of FLD2 * FLD1 as a D3 quantity would be 2.442. This result is rounded to FLD1 = 2.4.

If an internal result exceeds the maximum precision of any factor in the expression, it is rounded. For example, given FLD1/D1, FLD3/D3, FLD4/D4, and FLD5/D5 and the expression FLD1 = FLD3 * FLD4 * FLD5, the result of the first multiplication (FLD3 * FLD4) is accurate to 7 places. But the most precise factor is FLD5, so the intermediate result is rounded to 5 places before being multiplied by FLD5. This reduces the possibility of overflow in intermediate results (too many digits for a Ln, Dn or Fn field to hold).

If FLD2/D2 equals 3.00 and FLD1/D1 equals 2.0, then in the expression FLD1 = FLD1 / FLD2, the internal result of FLD1 / FLD2 as a D2 quantity is 0.67. It is rounded to FLD1 = 0.7. Intermediate rounding to the maximum accuracy of any factor works as in multiplication.

Note that divide and truncate (//) truncates the internal result. FLD1 = FLD1 // FLD2 produces the internal result 0.66, which is truncated to FLD1 = 0.6 in the result.

One final point: intermediate results are kept in L, D or F fields, and can therefore "overflow" out of these fields. As a rule of thumb, if your application does complex computations and you require 10 digits of precision, or more, use F fields.

8.3 Comparison and Special Operators

The comparison operators used in ADMINS are:

Operator	Function
EQ	equal to
NE	not equal to
GT	greater than
LT	less than
GE	greater than or equal to
LE	less than or equal to
BET	between
INCL	find string constant in An (alpha) field

For example:

```
A EQ B
C GT D
E GE 50.00
```

Comparison operators work on all data types. However, field types cannot be mixed in comparison expressions. Alphanumeric ("An") fields **must** be of the same width to be compared. For example, given an expression of the form:

```
IF AFLD EQ BFLD THEN ....
```

If AFLD is a field of type "A10", then BFLD must be of type "A10".

The syntax for between (BET) is A BET B AND C, where C is greater than B, as in:

```
DATE BET 01-JAN-75 AND 31-DEC-75
```

A special syntax form is provided for use when EQ or NE comparisons are made to a list of values:

```
A EQ B OR C OR D ...
```

is the same as

```
(A EQ B) OR (A EQ C) OR (A EQ D) ...
```

and

```
A NE B AND C AND D ...
```

is the same as

```
(A NE B) AND (A NE C) AND (A NE D) ...
```

The special operator **INCL** ("includes") works on alphanumeric fields only. It is used to find the **position** of an alphanumeric constant[2] in an alphanumeric field. (If the constant does not occur in the field, the INCL operation returns a value of zero.)

The syntax for INCL is:[3]

```
J = FIELD INCL 'constant'              (assignment statement)

IF (FIELD INCL 'constant') GT '0/I'   (conditional expression)
     THEN...
```

where J is an integer result field, and FIELD is the name of an alphanumeric field. For example:

```
...
J/I
...
J = NAME INCL 'DAVID'

    or

...
IF (NAME INCL 'Mr.') EQ '1/I' THEN GOTO STR_NAME END
...
```

A more general method for locating strings within strings is the LOCSTR subroutine described in Appendix H.5.6 “LOCSTR - Locate a String Within a String”.

8.3.1 WHILE Statements in RMOs

A while … endw statement has been added to the ADMINS RMO language. The syntax is:

```
WHILE expression THEN ;
    RMO statements ;
    …
ENDW
```

where expression can be any of the expressions used in an IF statement.

A typical use of the WHILE statement is:

```
WHILE X LT Y THEN ;
   Do something ;
   X = X + 1 ;
   ENDW
```

A **WHILE** statement must be contained within the same paragraph (i.e. between the **WHILE** and the **ENDW** no statement may start in column one (same rule as for IF and **END** statements).

---

2. The string that occurs after the INCL operator is always interpreted as a constant, even if it is not enclosed in quotes.
3. Conditional expressions (i.e. IF, SELECT etc.) that use INCL must explicitly type a constant that follows the comparison operator as an integer (e.g. '0/I'). Otherwise the compiler, following the "look to the immediate left" rule (see Section 8.1 “Constants”), will return a field type mismatch.

A down to earth use of the **WHILE** statement would be:

```
WHILE YEARS LT ADULT THEN ;
   GOSUB DO_SCHOOL ;
   YEARS = YEARS + 1 ;
   ENDW
```

Before WHILE statements this would have to be written something like:

```
WHILE: IF YEARS GE ADULT THEN GOTO IS_ADULT END
       GOSUB DO_SCHOOL  ;
       YEARS = YEARS + 1  ;
       GOTO WHILE ;
       END :
IS_ADULT: Continue other statements.
```

8.4 Logical Operators

The operators AND, OR, and NOT are provided for connecting comparison expressions to make Boolean expressions.

Examples

```
DATE GT 01-JAN-75 AND AMT NE 0

(ACCT# BET A0000 AND A9999) OR AMOUNT / 2. GT 100.00

GROSS NE 0 AND (DEDUCT EQ 1 OR (TAXCD EQ 1 OR 2))

NOT (A EQ B)
```

8.5 Conditional Statements

The IF_THEN_ELSE_END structure is used to conditionally compute a result. (The more broad use of conditional statements in record maintenance procedures is discussed in Section 9.6 “PROGRAM Section”). The IF_THEN_ELSE_END is **not** used in expressions which are explicitly conditional, such as the SELECT statement in the file definition, Message fields and Check statements in the screen instruction file, and SELECT and RECODE statement in reports. Rather the conditional structure is used in expressions which are expected to yield a value result, such as the virtual field computation in the screen instruction file and the CREATE statement in the report. For example:

```
V TAXDUE/D2 IF TAX LE 50.00 THEN TAX ELSE :
  TAX / 2. END

CREATE MSG/A7 IF THISDATE GT DUEDATE THEN 'Overdue' :
  ELSE ' ' END
```

When IF statements are nested only one terminal END is used. (This is not so for record maintenance procedures as is described in Section 9.6.1 "Record Maintenance Paragraphs".) For example:

```
V TAX/D2 IF XELD EQ 0 THEN GROSS * RATE ELSE :
  IF CB LT GROSS THEN CB * RATE ELSE GROSS * FROZRATE END
```

8.6 Parentheses

Parentheses are used to indicate precedence of expression evaluation. For example:

```
D = A * B + C
```

could mean multiply and add or add and multiply. Following the general rules of expression evaluation the multiplication precedes the addition.

ADMINS evaluates operators in expressions from left to right with the precedence determined as follows:

```
(1)  *   /   //  %
(2)  +   -
(3)  INCL
(4)  EQ   NE   GT   LT   LE   GE   BET
(5)  NOT
(6)  AND   OR
 (7)  IF_THEN_ELSE_END
```

For example, multiply and divide have equal precedence but precede add and subtract. Use parentheses to make the order of evaluation explicit. In the above example, if the user wanted the addition to precede the multiplication the expression would be written as follows:

```
D = A * (B + C)
```

8.7 Arrays

Local fields can be treated as arrays in record maintenance procedures. This feature is exclusively used in record maintenance procedures, and therefore is described in Section 9.5.1 "Creating Local Fields". However, there is also the ability to treat actual fields from the DEF as arrays. This can be used in expressions in CMP, REPORT, and SCREEN. In this facility the subscript is used to pick out a field from the file definition with respect to the field which is the base of the array. For example, consider the following DEF:

```
*
MAS 1000
*
ACCT# X999999 KEY1
DESCRIP A30
JAN D2
FEB D2
MAR D2
APR D2
MAY D2
JUN D2
JUL D2
AUG D2
SEP D2
OCT D2
NOV D2
DEC D2
```

The following expressions are valid, where "J" is of type I (integer).

```
AMT/D2 = JAN(0) + JAN(1)

J = 0
NXJ: TOTAL = TOTAL + JAN(J) ; J = J + 1 ; IF J LE 11 THEN
    GOTO NXJ END
```

In the first example AMT would contain the sum of the JAN and FEB amounts. In the latter example, which could only appear in a record maintenance procedure (RMS), TOTAL would contain the sum of all 12 monthly amounts.

A subscript of zero selects the array base itself. Negative subscripts are also permitted. The only restriction on subscripting is that the result field (after evaluating the array subscript) must fall within the DEF fields and be of the same type as the field at the base of the array. Also the subscript itself must be of type integer. Array notation on the DEF field names is supported in REPORT, CMP and SCREEN. (Note that in array expressions using fields from the file definition, the initial field is selected by a subscript of zero. In the array notation which operates on **local array fields** created in record maintenance procedures (RMS), the initial element of a local array is selected by a subscript of one.)

The apparent restriction that the base of the array and the result fields after subscript evaluation must be of the same type does not preclude "two dimensional" or "multi-dimensional" array effects. For example, consider the following DEF.

```
*
MAS 10000
*
ACCT# X99999 KEY1
TOTAL D
1TYPE A1
1AMT D
2TYPE A1
2AMT D
3TYPE A1
3AMT D
4TYPE A1
4AMT D
5TYPE A1
5AMT D
```

If we wanted to write a record maintenance procedure that accumulated in TOTAL all the amounts of type 'P'.

```
J = 0
LOOP: IF 1TYPE(J) NE 'P' THEN GOTO NEXT END
TOTAL = TOTAL + 1AMT(J)
NEXT: J = J + 2 ; IF J LE 8 THEN GOTO LOOP END
```

Notice, we increment the array subscript, J, by 2 each time through the loop, to move ahead two fields in the DEF, i.e. to the next nTYPE and nAMT fields.

8.8 Subroutines

ADMINS has a subroutine library to support many functions, some referred to in this Section, such as NCAT for converting between field types and STR for extracting part of a string. A complete description of all subroutines is included in Appendix H: "Subroutines".

Chapter 9:CMP: The Record Maintenance Compiler

CMP is the compiler for record maintenance procedures. A record maintenance procedure is a program that acts on and can manipulate the records in an ADMINS data file. A record maintenance procedure is prepared by entering (usually via a text editor) the statements of the procedure into an instruction file with a file type of ".RMS", e.g. "NAME.RMS". The compiled version of a record maintenance procedure, prepared by CMP, is stored in a record maintenance object file with a file name the same as the RMS and a file type of ".RMO", e.g. "NAME.RMO". If the logical name ADM$OBJECT is assigned the RMO is placed in the directory ADM$OBJECT otherwise it is placed in the same directory as the RMS. The RMO is then available for use with the MAINT, TRANS, PROD, and REPORT commands.

9.1 CMP Dialogue

Compile RMS by entering CMP at the system prompt. The file name may be included on the command line or CMP will prompt for the name. If the compile is successful, CMP will print a message saying the RMO was written, the number of paragraphs in the RMS, the total size of the object file and the size of the constants used in the RMS. If the compile is unsuccessful, CMP will print an error message and terminate. For example, given an instruction file called "NAME.RMS":

```
$ cmp name
NAME.RMO WRITTEN. 2 PARAGRAPHS. OBJ-SIZE: 53  CONS-SIZE: 11
```

or

```
$ cmp
PROG NAME:name
NAME.RMO WRITTEN. 2 PARAGRAPHS. OBJ-SIZE: 53  CONS-SIZE: 11
```

9.2 Outline of A Record Maintenance Procedure (RMS)

The outline of a record maintenance procedure instruction file (RMS) is as follows:

```
FILE file-name

TABLE table-file-name [local-name(s)]

LOCAL

name/type [value]
name/type(n) [value1 value2 etc.]
...

PROGRAM

executable statements
...
```

9.3 FILE Statement

The FILE statement names the file on which the record maintenance procedure will operate. For example:

```
FILE PAYROLL.MAS
```

9.4 TABLE Statement

The TABLE statement causes CMP to create local arrays and load these arrays with the values from the TABLE file. The names and types of the local array fields are taken from the names and types of the fields in the TABLE file. TABLE statements are described in Section 9.8 "TABLE Statement".

9.5 LOCAL Section

The LOCAL statement begins a section that includes "local" field names. A local field is a field that is not stored in the record but is maintained in memory and is usable in expressions along with constants and actual field names from the record. Local fields are **not initialized for each new record** that the record maintenance procedure receives for execution. Consequently local fields can be used to hold values that in some way relate the individual records of the file being processed, each to the next. For example, use a local field to compare a particular field with the same field in the next record.

9.5.1 Creating Local Fields

The definition of the local fields follows the LOCAL statement. Each local field has one of the following formats:

```
NAME/TYPE [value]

NAME/TYPE(n) [value1 value2 etc.]
```

The first format defines a local field by naming it, assigning it one of the ADMINS data types, and, optionally, initializing it with a value. For example:

```
AMOUNT/D
RATE/D2 2.75
YEAR/I 75
TODAY/DA 21-SEP-75
NAME/A20
#BILL/XA99999 M00000
```

The second format is used to define local arrays. Arrays are generally used for short tables. The "(n)" is the number of elements to be stored in the array. The contents of the array can be initialized by values on the line, or alternately the values can be set during record maintenance execution. The subscript value "1" selects the initial value in a local array. An example of a local array would be as follows:

```
RATES/D2(5) 1.75 2.50 3.75 5.00 7.50
```

Using the above local array statement, "RATES(3)" would equal "3.75".

A reference to a Data Dictionary element may be substituted for the field type specification as described in Section 1.3.5 “Referencing Data Dictionary Elements”, e.g.

```
RATES/@XRATE(5) 1.75 2.50 3.75 5.00 7.50
```

9.5.2 Checking the Subscript Value for Local Arrays

ADMINS checks for local array subscripts that are out of bounds: that is, at run time, the ADMINS image that is calling the RMO will prevent it from using a subscript value that would point beyond either end of a local array. If this situation is detected, ADMINS displays a message (xec011) giving the name and size of the array and the value of the offending subscript, and exits with fatal error status.

The error in the RMS must be corrected. If necessary, the specific RMO paragraph that causes the problem can be located by running the application in test mode until it gives the "xec011" message and exits.

Array subscript checking applies only to fields that are declared as local arrays in the RMS, including local arrays created by TABLE statements (see Section 9.8 "TABLE Statement"). It does not apply to the subscript (offset) notation that can be used with repeating sets of fields in the main file (see Section 8.7 "Arrays") which are checked for validity by other means.

Run time array subscript checking is also performed when subscript notation is used with global (G$) fields (see Section 5.5.9 "Global Fields"). When a subscript is detected that attempts to reference beyond either end of the global record, ADMINS displays a message ("xec012" for global fields) and exits with fatal error status.

In an RMS these LOCAL field declarations:

```
MYFIELD/D
MYFILD/D(1)
```

are equivalent. They both declare the scalar variable MYFIELD, not an array. The informational message:

```
cmp979 A dimension of one (1) does not declare an array
```

followed by the source line and file name will be displayed by AdmCmp if it encounters a declaration that uses array notation but attempts to declare an array size of "1".

AdmCmp also checks and notes array notation when it is used to reference scalar fields in the PROGRAM section of the RMS, e.g. if MYFIELD is declared as above (either with or without array notation) and this statement was encountered in the PROGRAM section:

```
MYFIELD(I) = 7
```

AdmCmp will then display the error message:

```
cmp978 Dimension not allowed for scalar variable
```

followed by the source line and the file name, and error exit without completing the compilation.

There are a number of exceptions to this rule:

1. If the field is part of the DEF for the file it is assumed you use subscripting to access one of a series of similarly defined fields in the record (e.g. BUDGET01/D - BUDGET12/D).
2. If the field is a global or shared global field (the field name starts with G$ or SH$) subscription of scalar variables are allowed.
3. If the field is from a TABLE statement.
4. For compatibility reasons AdmCmp will allow subscripting of a scalar variable with the constant 1 (one), e.g. MYFIELD(1), since this syntax will reference the one and only occurrence of the data for this field. It will issue the cmp978

message as a warning, but will continue the compilation. If the subscript is anything but (1) (e.g. another constant, or a field name) the message cmp978 is issued and the compilation aborts.

9.5.3 ALIAS: Create Field Names for Local Array Elements

The ALIAS statement automatically creates a series of distinct local field names (or "aliases") for the elements of local arrays. ALIAS provides an easy way to refer to these array elements in situations where array notation cannot be used. For example, if you want to reference an array element in the DETAIL layout for a report, or in the APPEND paragraph of a TRS. Place the ALIAS statement in the LOCAL section of the RMS, using the following syntax:

```
ALIAS ARRAY_NAME[(RANGE)]  [...]
```

where ARRAY_NAME is the name of a local array declared previously in the RMO, e.g.:

```
ITEM/X9999(50)
ALIAS ITEM
```

RANGE is optional. If RANGE is not present, ALIAS will create names for every element in the array. In the example above, ALIAS would generate field names ITEM_1 through ITEM_50 that can be used to refer to elements 1 through 50 of the local array ITEM.

Use RANGE to indicate where in the array to start and stop creating alias field names. RANGE can be specified in two ways:

```
array_name(START) or  array_name(START:END)
```

where START and END are numeric constants. START by itself specifies where in the array to begin generating field names, while the START:END syntax specifies the range of array elements for which names are to be generated, e.g.:

```
ITEM/X9999(50)
ALIAS ITEM(26)
```

would begin creating names starting with ITEM_26 (for element 26) and continue through to ITEM_50 (for element 50, the last element of the array).

```
ITEM/X9999(50)
ALIAS ITEM(26:30)
```

would begin creating names starting with ITEM_26 (for element 26) and continue to ITEM_30 (for element 30).

Local fields created by ALIAS can be used like any other local fields, with one significant difference: **there is only one copy of the data.** The ALIAS field is just **a second name** for an array element. If the RMO changes AR(3), the ALIAS field AR_3 immediately reflects the change; and, likewise, changing AR_3 causes an immediate change in AR(3), because AR_3 and AR(3) are just two ways of referring to the same data.

Since field names cannot be over 18 characters long, ALIAS may not work with a long array name. Array names of 12 characters or less will always work with ALIAS.[1]

ALIAS will not work for array names that begin with a non-alphabetic character, because adding the numeric character suffix would result in an invalid field name (see Section 2.4.1 “Field Names”).

Note: fields created with ALIAS count toward the limit of 1000 fields in a virtual record. So don't use ALIAS unless you need it; and don't ALIAS more array elements than you need.

1. No local array size can be more than a 5-digit decimal number, so a 12-character array name, plus an underscore, plus an array subscript up to 5 characters long will always be 18 characters long or less and thus always work.

9.5.3.1 Example: Using ALIAS with REPORT

An RMO written for use with REPORT

```
LOCAL
...
AR/I(10)
ALIAS AR
...
```

ALIAS generates field names AR_1 through AR_10 that can be referenced in REPORT CREATE statements, printed, or used in any other way that a local field can be used. The REPORT instruction file might contain:

```
...
EXECUTE
...
CREATE EXTAMT/I AR_4 + 100
...
DETAIL
     AR_1---  AR_2---  AR_3---
...
```

9.6 PROGRAM Section

The PROGRAM statement precedes an executable portion of the record maintenance instruction file. Executable statements consist of arithmetic and Boolean expressions that are formed using the rules described in Chapter 8: "Expressions". In addition, the record maintenance compiler can compile particular statements unique to record maintenance.

The general expressions in ADMINS allow for constants, arithmetic operations (+ - * / //), comparison operations (EQ NE GT LT LE GE BET INCL), Boolean connectives (AND OR NOT), and IF_THEN_ELSE_END conditional structures. These facilities are described in Chapter 8: "Expressions". Subroutines may also be included in general expressions and are described in Appendix H: "Subroutines".

In addition to these more general facilities provided throughout ADMINS, record maintenance has its own operations for moving data from one field to another (=), GOTO, GOSUB, and STOP statements, local array subscripting, statement labels, the RET instruction, and the semicolon to separate multiple statements appearing in the same paragraph.

9.6.1 Record Maintenance Paragraphs

A record maintenance procedure consists of paragraphs of statements. A paragraph starts with a statement that begins in column one and consists of that statement plus all subsequent statements that are indented inward from column one, up to a non-indented statement that starts in column one (i.e. the beginning of the next paragraph), or up till the end of the program, whichever comes first. Paragraphs serve several purposes.

5. Any nesting of conditionals occurs within the same paragraph. Therefore the IF and END statements in a paragraph must balance.
6. The only statement in a paragraph that can be labeled is the **initial,** i.e. the non-indented, statement. Hence, every GOTO necessarily transfers control to the first statement in some paragraph.
7. During "test mode" (discussed with the MAINT command in Section 10.2 "Test Mode"), the instruction file statements are displayed on the screen a paragraph at a time. Hence a paragraph should be relatively small, never more than about seven or eight lines. Note, a line may contain several statements using the semicolon to separate individual statements on a line.

However, the end of a line does not terminate an individual statement. That is, statements can cross lines. The semicolon must be used to terminate statements even when the statement ends at the end of line. End of paragraph does automatically terminate a statement.

9.6.2 Record Maintenance Statements

1. The "=" operator is used to instruct movement of data from one field to another. The ";" is used to separate statements appearing in the same paragraph. For example:

```
DATE = 01-SEP-75

TOTAL = TOTAL + DETAIL

CWITH = CWITH + BWITH ; CFICA = CFICA + BFICA
```

2. The GOTO statement is used to transfer control to a labeled paragraph. For example:

```
GOTO FINIT

GOTO NEXT
```

3. The STOP statement is used to stop execution on the particular record. That is, when STOP is executed the current record is re-written to the disk file and the next record is read for processing. An implied STOP is at the end of a record maintenance instruction file.
4. Statement labels are used to label particular statements at the head of a paragraph so they can be transferred to via the GOTO AND GOSUB statements. For example:

```
START: IF IDENT BET P0000 AND P9999 THEN
 GOTO PURCHASE END
```

Labels may be up to 20 characters long.

5. Array subscripts are used to access a particular element of an array. The array subscript may be an actual integer field, a local integer field, or an integer constant. Arrays may be either local or actual fields from the DEF file. The former, local arrays, were described above. The latter, actual field arrays, are described in Section 8.7 "Arrays".

```
IF UNEXPA BET LO(J) AND HI(J) AND MARITL EQ MSTAT(J)
     THEN GOTO FOUND END
```

9.6.3 The GOSUB Statement

The GOSUB statement ("go to subroutine") enables an RMO to go to another paragraph, execute statements and then return to the statement following that GOSUB call. The GOSUB facility uses two reserved words which cannot be used as field names: GOSUB and RET.

Whenever GOSUB is invoked, the program goes to the indicated paragraph and executes statements until the instruction RET is encountered. When RET (for 'return') is executed, the RMO goes to the statement just after the last executed GOSUB and continues processing from that point.

Every RET is paired with the last executed GOSUB. GOSUBs can be nested up to 25 consecutive GOSUB calls without an intervening RET. One GOSUB can go to a paragraph which contains another GOSUB, which in turn can go to a paragraph with yet another GOSUB, etc. Each RET will return to the statement following its corresponding GOSUB call.[2]

The syntax in the program portion of the RMO is:

```
GOSUB statement-label
```

The statement-label is an RMO label as defined above in Section 9.6.2 "Record Maintenance Statements". The GOSUB statement is similar to the GOTO statement except for GOSUB's capability to be returned (via RET) to the statement following the GOSUB call.

To illustrate one use of GOSUB:

```
FILE INVITE.MAS
LOCAL
S$S/A6
M$M/A2
CODE/A2
WORD/A20
PROGRAM
IF S$S EQ 'CODE1' THEN
   CODE = CODE1 ; GOSUB STEP1 ; WORD1 = WORD ; GOTO DONE END
IF S$S EQ 'CODE2' THEN
   CODE = CODE2 ; GOSUB STEP1 ; WORD2 = WORD ; GOTO DONE END
IF S$S EQ 'CODE3' THEN
   CODE = CODE3 ; GOSUB STEP1 ; WORD3 = WORD ; GOTO DONE END
DONE: STOP
STEP1: IF CODE EQ 'C' THEN WORD = 'CREDIT' ; RET END
       IF CODE EQ 'D' THEN WORD = 'DEBIT' ; RET END
       IF CODE EQ 'U' THEN WORD = 'UNKNOWN' ; RET END
```

2. A paragraph reached via GOSUB can contain STOP. RET is not needed before STOP. All pending return destinations are discarded by STOP.

9.7 Parameterization

Any string in the record maintenance instruction file can be parameterized by placing the string in angle brackets. This means that the contents of the string, i.e. the text between the "<" and ">", is prompted on the user's terminal during compilation. The string typed in response to the prompt by the user is inserted into the instruction file **for the purpose of the particular compilation** in place of the parameter text. For example:

```
FILE <FILE NAME>

DATE/DA <CHECK DATE>

IF ACCT# BET <LO> AND <HIGH>
```

Once text has been supplied for a particular parameter, i.e. a particular angle bracketed string, then that text will be substituted for the parameter each time it is encountered.

If however, the parameter is enclosed in double brackets, as follows:

```
IF ORDATE NE '<<Select Date or Press Return for All>>' THEN GOTO
OUT END
```

and the user does not supply a response, then CMP will ignore the entire instruction line which contained the double bracketed string.

9.7.1 Logical Parameters

If the parameter string contained in the angle brackets begins with the characters “L_”, (e.g. <L_fieldname>), then AdmCMP first tries to translate the prompt as a logical name. If the logical name has been assigned in either the process, desktop, or system logical name tables, the user is not prompted for the contents of the parameter. Instead the value of the logical name is substituted for the prompt. Parameters which begin with the characters “L_” and are assigned as logical names are called "logical parameters".

When the logical names exist, the display of logical parameter prompts and their values can be suppressed by assigning the lowercase letter "c" to the logical name OPTION (see Appendix A: “Options”).

If a parameter beginning with “L_” is not assigned as a logical name, then the user is prompted for a value as in standard parameterization (see Section 9.7 “Parameterization”).

Prompting for values when the logical name is not assigned can be avoided entirely by supplying a default value in the parameter string, as follows:

```
<L_MINIMUM=0>
```

Specify the default value for the logical name by appending “=value” to the logical name inside the angle brackets. In the example above if the logical name L_MINIMUM is not assigned, the value “0” will be substituted for the parameter.

9.8 TABLE Statement

The TABLE statement in a record maintenance procedure[3] is used to read the contents of a modest sized ADMINS file into the local arrays of the record maintenance procedure. The field names from the "table file" then automatically become local array names in the record maintenance procedure. The TABLE statement in the following example will create local arrays for all the fields in WITHOLD.TAB:

```
FILE PAYROLL.MAS
TABLE WITHOLD.TAB
...
```

If a field name in the TABLE file is the same as a field name in the main RMO file CMP will exit with a diagnostic message. Consider the following payroll file and a withholding table definitions:

```
*     PAYROLL.DEF
MAS 1000
EMPL# X99999 KEY1
NAME A20
...
AMT D2
...

*     WITHOLD.DEF
TAB 100
MARITL A1
LOW D2
HIGH D2
AMT D2
PERCENT D2
```

Then attempting to compile an RMO that contains:

```
FILE PAYROLL.MAS
TABLE WITHOLD.TAB
...
```

would result as follows:

```
$ cmp deduct
cmp965 Field "AMT" in table file "WITHOLD.TAB" already present in
file "PAYROLL.MAS"
   Line 2: TABLE WITHOLD.TAB

cmp965 Explanation: Fields in a table file must either have
       different names than the fields in the file referenced by
       the FILE statement or they must be renamed on the TABLE
       statement.

       Reference: ADMINS Procedures Manual - 9.8

       User Action: Rename the table fields on the TABLE statement.
```

AMT is present in both the payroll master file and in the withholding table file.

3. Although TABLE in an RMS and TABLE in a REPORT both have the same general kind of purpose, i.e. accessing files as "tables", the two TABLE instructions are quite different in the detail of their operation.

To deal with field name conflicts TABLE statements have a syntax[4] to assign a new "local" names to the TABLE file's fields:

```
TABLE table-file-name local-name1 local-name2 etc.
```

In the example above we could use

```
TABLE WITHOLD.TAB MARITL LOW HIGH TABLEAMT
```

which assigns the new local name TABLEAMT to AMT for use in this record maintenance procedure. MARITL, LOW and HIGH are given the same names and are only present for syntactic purposes as place-holders to "space" up to TABLEAMT. PERCENT, the last field in the TABLE file, need not be present in the TABLE statement because it is not needed for place-holding and does not need to be renamed.

Tables are **not dynamic.** That is, the TABLE file is loaded into the RMO at compile time. **If the contents of a TABLE file changes the RMS must be re-compiled to include the new data.**

Use the ARSZ subroutine (see Appendix H.11.3 “ARSZ Subroutine”) to determine the dimension (size) of an array created by a TABLE statement, i.e. the number of records in the TABLE file. This information is necessary, for example, to control when looping through the elements of an array should stop.

9.9 Declaring Local Fields in Indirect References

In order to write generalized indirectly-referenced[5] file "subroutines" (or "macros" or "modules") you need a way to declare local variables within the "@@file" which will be included in the RMS at compilation. To accomplish this, place a LOCAL statement and a list of the (additional) local field declarations before the PROGRAM statement inside the "@@file" (local fields must be declared before they are used in any procedural code).[6]

4. Use the colon continuation convention to extend the list of field names beyond one line.
5. See Section 1.3.3.1 “Passing Parameters in Indirect References”
6. LOCAL/PROGRAM pairs are not limited to indirectly referenced program fragments, but may be used throughout the PROGRAM section of the main .RMS file. This might improve the readability of RMS files which have many local fields that are used only in specific parts of the program: such local fields could be declared in separate LOCAL sections just above the PROGRAM segments where they are used.

The following example illustrates the syntax:

```
* GETDEPT.FIL: Routine to translate D$EPT logical name
*
PARAMETER FLD1
*
LOCAL
*
STAT/I
LNAM/A20 'D$EPT'
FLD1/A40
*
PROGRAM
*
G_FNM: STAT = TRLOG(LNAM,FLD1)
IF STAT LE 0 THEN ;
   .
   .
```

The .RMS file could include this section of code with an indirect reference:

```
@@GETDEPT.FIL DEPT
```

(i.e. load the translation of logical name D$EPT into field DEPT) which would be compiled as if the RMS contained the following code:[7]

```
*
LOCAL
*
STAT/I
LNAM/A20 'D$EPT'
DEPT/A40
*
PROGRAM
*
G_FNM: STAT = TRLOG(LNAM,DEPT)
IF STAT LE 0 THEN ;
   .
   .
```

Fields declared in the LOCAL section of an indirect reference file are compiled by CMP like any other local fields, and are available to the entire .RMS program. Therefore, local fields declared in "@@" files should use a field-naming convention ensure that they will not duplicate field names elsewhere in the RMS or in other include files referenced in the same compilation. In particular, if an "@@" file declares local variables and is referenced more than once in the same compilation, care should be taken to declare **all** local variables as parameters, because, as always, a local field can be declared only once (otherwise, CMP will exit with a "field already exists" message).

7. As is described in Section 1.3.3.1 “Passing Parameters in Indirect References”, each occurrence of a string that appears in the PARAMETER statement of the indirectly referenced file is replaced with the corresponding parameter value given on the "@@filename" line, before it is read as an instruction by CMP.

9.10 Record Maintenance Examples

The following procedure operates on a file of payments sorted by vendor number (#VEND). The procedure assigns a check number (CK#) to each payment. Each payment to the same vendor is assigned the same check number. The initial check number is parameterized.

```
* Assign check numbers
FILE PAYM.MAS
LOCAL
SEQ/I <STARTING CHECK NUMBER>
LASTVEND/X9999 0000
PROGRAM
IF #VEND EQ LASTVEND OR LASTVEND EQ 0000 THEN GOTO SAME END
SEQ  = SEQ + 1
SAME: CK#  = SEQ ; LASTVEND = #VEND
```

The following procedure reads a withholding table from the file WITH.TAB and computes the withholding amount. The fields in WITH.TAB are MSTAT, LO, HI, AMT and PC. The rule for using the table is if the total pay lies between a LO and HI entry and the persons marital status is equal to MSTAT for that entry, then the withheld amount is AMT plus PC times the total pay minus the LO, all taken from that table entry. The last record in the withholding table file contains an AMT of -1.

```
* Compute withholding. Store it in WITH.
FILE PAYROL.MAS
TABLE WITH.TAB
LOCAL
J/I
PROGRAM
J  = 1; WITH  = 0.00
NEXT: IF MSTAT(J) EQ MARITL AND TOTPA BET LO(J) AND HI(J)
  THEN WITH  = AMT(J) + (TOTPA - LO(J)) * PC(J) ; GOTO FINIT
  ELSE IF AMT(J) NE -1 THEN J = J + 1 ;
  GOTO NEXT END END
FINIT: STOP
```

A file contains dates in a DATE field. These dates are days in July or August. Each record is to be coded to show the number of the week in which its date lies.

```
* Recode DATE to WEEKNO
FILE ACTVTY.MAS
LOCAL
J/I
MON/DA(8) 03-JUL-75 10-JUL-75 17-JUL-75 24-JUL-75
  02-AUG-75 09-AUG-75 16-AUG-75 23-AUG-75
SUN/DA(8) 09-JUL-75 16-JUL-75 23-JUL-75 30-JUL-75
  08-AUG-75 15-AUG-75 22-AUG-75 30-AUG-75
PROGRAM
WEEKNO  = 0 ; J  = 1
NEXT: IF DATE BET MON(J) AND SUN(J) THEN WEEKNO  = J ; STOP
  ELSE J  = J + 1 ; IF J LE 8 THEN GOTO NEXT END END
```

An accounting file contains an eight digit account number with fund (2 digits), department (3 digits), and object (3 digits) parts. Object codes in the range from 200 to 299 in department 416 are to be re-filed under department 616. The NCAT and STR subroutines described in Appendix H.3 “Concatenation Subroutines” and Appendix H.5.1 “STR - Select Part of a Field” are used.

```
*  CHANGE DEPT FROM 416 TO 616 FOR OBJECTS 200 TO 299
FILE STAT.MAS
LOCAL
FUND/X99
DEPT/X999
OBJ/X999
PROGRAM
FUND = STR(FUND,ACCT,'1/I','2/I') ;
  DEPT = STR(DEPT,ACCT,'3/I','5/I') ;
  OBJ = STR(OBJ,ACCT,'6/I','8/I')
IF DEPT NE 416 OR OBJ LT 200 OR OBJ GT 299
  THEN STOP END
DEPT = 616 ; ACCT = NCAT(ACCT,FUND,DEPT,OBJ)
```

Note that in the last example, if ACCT is the key to the file STAT.MAS, then STAT.MAS must be sorted in order to access the changed accounts by their new number.

9.11 DEBUG Mode

Commands that call an RMO can run the RMO in **Debug Mode.** Debug Mode allows breakpoints and watchpoints to be set in the RMO when it is running, statement by statement execution of the code, examination and changing of current values of variables, and detailed examination of the environment where the RMO is running.

To use debug mode, the RMS must first be compiled with the **-ADBG** option[8]:

```
cmp rms_name -ADBG
```

To enter debug mode, the logical name **ADM$DEBUG_RMO** must be assigned, and the command's "Test Mode" must be enabled (if debug mode is active, it disables and replaces test mode).[9]

8. The -ADBG option must come after the name of the RMS. -ADBG makes the RMO larger, which slows down program initialization at run time, so once the program is debugged, it should be recompiled without the -ADBG option.
9. Test mode does not have to be enabled in TRANS. For example, in MAINT test mode is enabled by replying "Y" to the "Test mode? (Y/N)" prompt.

9.11.1 Source Code Window

When the RMO is called with debug mode active, a listing of the source code is displayed, with possible breakpoints indicated by video highlighting[10] of the first character in the statement (breakpoints can be at the start of any statement.). Breakpoints are places in the code that you can designate for the debugger to suspend execution of the program before processing.

Watchpoints can be set only where the start of the statement is a field receiving a value (e.g. FIELD = ...)[11]. Watchpoints are field names for which the debugger is to suspend program execution after they are written to by the RMO[12]. The debugger displays the value the field had before being written by the RMO, and the value after being written[13] Program execution is suspended immediately after the statement that changed the watchpoint field.

10. By default, video highlighting is "bold". To specify another kind of highlighting, assign one of the following codes to ADM$DEBUG_RMO:
    R - for Reverse video
    U - for Underline
    B - for Blink
    S - for Bold (the default)
    any other character assigned ADM$DEBUG_MODE will result in default highlighting of possible breakpoints.
11. You cannot set a watchpoint at an array variable if the subscript is a variable (e.g. AR(3) is OK, but AR(N) is not)
12. Debugger watchpoints will not detect changes in fields made outside the RMO, e.g. by moving to a new record, by typing in TRANS, etc. You may, however, at any time use the Print command to print the current value of any field in the virtual record. If you need to set a watchpoint on a variable not being set by the RMO, insert a statement like:

```
DUMMY = VARIABLE
```

    at an appropriate point in the RMO (e.g. at S$S EQ 'BEGREC'), and set a watchpoint on DUMMY (remember to declare DUMMY in the LOCAL section).
13. The before and after values at a watchpoint may be the same, as the debugger only checks that the RMO has written to a field, not if the value actually changed.

Keystroke commands available in the source code window are:

Keystroke	Action
Arrow keys	Move between statements/possible breakpoints
PREV/NEXT	Move between pages
B	Toggle: set/cancel a Breakpoint at cursor
W	Toggle: set/cancel a Watchpoint at cursor
N	Find Next breakpoint
P	Find Previous breakpoint
T	Go to Top of source
E	Go to End of source
S	Enter Search string (terminate with Ctrl/R to search backwards)
R	Reverse search direction
C	Continue search
G	Continue execution
H or HELP	Source code window help
Ctrl/W	Refresh source code window

9.11.2 Command Line

After you have set any desired breakpoints and watchpoints, type "G" to start program execution. Communication with the debugger resumes in the command line window, at the **ADBG>** prompt: The following commands are available:

Command	Action
S[tep]	Step to next statement (execute current statement)
G[o]	Continue execution to next breakpoint or next watchpoint change. If you type 'Go' at the ADBG> prompt without having to set a breakpoint or watchpoint, your RMO will run to completion without stopping the debug windows again.
N[ext]	Continue execution to next paragraph
SO[urce]	Go to source code window to set/cancel breakpoints/ watchpoints
H[elp] or HELP	Help
P[rint] FIELDNAME [FIELDNAME…]	Display values of field(s). Prints current value of a field or all elements of an array. ADBG> P XAR(3) will print the value of element 3 of array XAR, and ADBG> P XAR(2:6) will print the value of elements 2 through 6 of array XAR.

Command	Action
W[atch] FIELDNAME[N[:M] [FIELDNAME[N[:M]…]	Set/cancel watchpoint(s). FIELDNAME may be any field or array name manipulated by the RMO. Array elements (i.e. AR(4) or AR(1:5)) are valid watchpoints: AR(4) sets a watchpoint on the 4th element in array AR, and AR(1:5) sets watchpoints on elements 1 through 5of array AR).
W[atch]	Display watchpoints
A[ssign] FIELDNAME VALUE	Put a VALUE into FIELDNAME
WH[atis] FIELDNAME [FIELDNAME…]	Show field type and dimension
C[ancel] W[atchpoints]	Cancel all watchpoints
C[ancel] B[reakpoints]	Cancel all breakpoints
Ctrl/W	Refresh window
Q[uit]	Quit execution

Chapter 10:MAINT: The Record Maintenance Processor

The record maintenance processor, MAINT, is used to execute procedures (RMO's) compiled by CMP. MAINT runs the procedure on the file designated by the FILE statement in the record maintenance procedure instruction file.

10.1 MAINT Dialogue

When the user calls MAINT the following dialogue ensues using "NAME.RMO" as the example:

```
$ maint
RECORD MAINT. NAME:name
TEST? (Y OR N):n
OPERATING ON NAME.MAS
OK TO CONTINUE?y
11:53:06.60
************************************************************
311 RECORDS PROCESSED   11:53:26.00
$
```

The "RECORD MAINT. NAME" may also be included on the command line as follows:

```
$ maint name
TEST? (Y OR N):n
OPERATING ON NAME.MAS
...
```

MAINT will execute the procedure called NAME.RMO. This is the file produced by CMP from NAME.RMS. If the user is not running in test mode, that is "N" was entered to the "TEST?" prompt, then MAINT performs the following steps.

1. MAINT reads an input record from the file named in the FILE statement.
2. MAINT executes the procedure.
3. MAINT writes the record back to the file.
4. If the file still has records to be processed then MAINT continues at step (1).
5. The file is closed and MAINT prints the "number of RECORDS PROCESSED" message.

When not in test mode, MAINT prints a line of asterisks to show the on-line user the progress through the file. The user can type "NO *" to the initial prompt to inhibit the line of asterisks.[1]

1. If the character "*" (asterisk) is included in the string assigned to the logical name OPTION, the printing of the line of asterisks to show progress through a file is suppressed in all ADMINS "batch" commands. See Appendix A: “Options”.

10.1.1 ADM$RECORDLOCK

If the RMO run by MAINT contains a field called ADM$RECORDLOCK/I, the RMO gets a call with ADM$RECORDLOCK set to 1 if the next record is locked.

The only possible actions at this call is to set ADM$RECORDLOCK to one of the following values:

```
1 = Wait for record
8 = Ignore record lock (get the record in read only mode)
16 = Skip the record
```

If the RMO does anything else at this point, the results are unpredictable. For example, if ADM$RECORDLOCK is present, the first statement in the PROGRAM section should be:

```
IF ADM$RECORDLOCK EQ 1 THEN
ADM$RECORDLOCK = desired action ;
STOP  ;
END
```

10.2 Test Mode

MAINT provides a test mode for debugging an RMO. It is recommended that every RMO be run in test mode to the satisfaction of the user before using the RMO in non-test mode on a production basis. As a matter of fact it is good practice to test run a few records before each production run of a fully tested RMO. This is especially good practice if there are several related RMOs, where one sets up conditions in the fields of the records for the others to use. The test run of a few records will show the user that in fact the proper set up preliminary RMOs have been run.

As should be evident from the previous paragraph the major purpose of test mode is to make execution of an RMO as **visible** as possible. In test mode each paragraph of an RMO is displayed on the screen[2] as it is executed. As well, the values of all fields in the paragraph are also displayed. The values are displayed in two states: before execution of the statements in the paragraph and after execution of these statements.

2. A video terminal is required to use test mode.

10.2.1 Test Mode Display Examples

```
                    RECORD: 5   STATEMENT: 1
SEQ = SEQ + 1 ; J = J + 1 ; TOTAL = 0
FIELD NAME     BEFORE                    AFTER

SEQ             4                        5
J               15                       16
TOTAL           13,768                   0

                    RECORD: 14   STATEMENT: 8
IF (TOTPA BET LO(J) AND HI(J)) AND MSTAT(J) EQ
  MARITL THEN WITH = AMT(J) + ( TOTPA -
  LO(J) ) * PC(J) ; STOP END
FIELD NAME     BEFORE                    AFTER

TOTPA           640.00                   640.00
J               7                        7
LO(J)           600.00                   600.00
HI(J)           700.00                   700.00
MSTAT(J)        M                        M
MARITL          M                        M
WITH            .00                      58.00
AMT(J)          50.00                    50.00
PC(J)           .20                      .20
```

10.2.2 Test Mode Operation

During test mode records are **not written back to the file.** Therefore test mode never changes any data values on the disk file.

When MAINT is entered in test mode the first record is read into memory, the first paragraph of the RMO is displayed on the screen along with the before and after values, and then MAINT awaits one of the following keystrokes.

1 to 9	The display of that number of paragraphs is skipped in the execution of the RMO on the current record. Note the skipped paragraphs are **executed**, but not displayed. This feature can be used to rapidly step through a repetitive loop.
R	The letter "R" causes MAINT to skip to the next record. The remaining paragraphs are not executed on the current record. MAINT continues by displaying the first paragraph on the next record.
S	The letter "S" causes MAINT to re-start the RMO at its first paragraph on the current record.
KEY (PF2)	In response to the KEY keystroke (PF2) MAINT prompts with "TYPE:" at the bottom of the screen. The user then enters a key value for a particular record in the file and presses "enter". If the records in this file have multiple keys then MAINT lets the user enter values for each key field. When MAINT has all the key value(s) necessary for searching out a particular record in the file, MAINT then finds that record, and continues by displaying the first paragraph of the RMO on that record. (If the record is not found, the nearest record, in sort order, is used.) This feature is particularly useful after an RMO is thought to be debugged but some records are still not coming out completely correct.
Ctrl/b	MAINT closes the active file and terminates.

Any other keystroke.	MAINT continues with the next paragraph on the current record. MAINT steps through the execution of the RMO on the data while the user watches. This stepping through proceeds at a pace controlled by the user pressing any keystroke.

10.2.3 Test Mode Hints

1. Keep the RMO paragraph sufficiently small so that the paragraph plus the list of field names will not require more than the 24 lines of displayable space on the screen. When the displayable space is exceeded MAINT will display as many fields as possible, truncating the display of the paragraph.
2. If one wishes to display the values for fields that are not used in the paragraph, then include these field names in the paragraph in a way that does not effect the execution of the statements in the paragraph. For example, to include the display of an account number in a paragraph that initializes a total value to zero, one might have the following:

```
ACCT# = ACCT# ; TOTAL = 0
```

10.3 Operate on KEY Values

MAINT can be instructed to operate only on those records in a file specified by their key value:

```
$ maint calc key
OPERATING ON PERS.MAS
OK TO CONTINUE?y
17:07:14.78

KEY:
```

The feature is invoked by placing the word "KEY" after the name of the RMO to be executed on the command line as shown. This causes MAINT to prompt for a key value. If the key in the file has multiple fields then the key is specified as a set of key values, one per field, separated by a blank. If only part of a key is entered then all records that have that partial key will be executed. The line of asterisks is suppressed when the KEY feature is used.

MAINT then runs the RMO on the records requested via the key value. When these records have been processed MAINT then prompts for another key value, and so on, until the user answers the prompt for a key with a carriage return.

10.3.1 Operate on Key Range

MAINT can be instructed to operate only on those records in a file specified by a range of key values:

```
$ maint
RECORD MAINT. NAME:name
TEST? (Y OR N):k
OPERATING ON NAME.MAS
LOW KEY VALUE: A
HIGH KEY VALUE: D
OK TO CONTINUE?y
11:53:06.60
************************************************************
311 RECORDS PROCESSED   11:53:26.00
$
```

The feature is invoked by responding to the "TEST? (Y or N):" test mode prompt with the letter "K" (for Key Range). This response causes MAINT to prompt for the range specification, "LOW KEY VALUE:" and "HIGH KEY VALUE:". If the key in the file has multiple fields enter values for some or all of the multiple keys, separated by blanks. Null values are used for minor keys not entered.

MAINT will accept logical names for the LOW KEY VALUE and HIGH KEY VALUE prompts. If the response either prompt begins with the letters "L$" then MAINT will attempt to translate the response as a logical name. If such a logical name exists, MAINT will use the string assigned to it as the (low or high) key value, otherwise MAINT will use the response directly as it usually does.

When the range specification is complete, MAINT prompts for confirmation:

```
OK TO CONTINUE?
```

If the response is "Y" MAINT then runs the RMO on the records within the specified range.

10.4 Controlling Write Back: W$W

When not in test mode MAINT writes each record back to the disk after the record has been processed. However, it is possible that the RMO only alters records selectively. In this case only the altered records need be written back to the disk. There is a way that the user can control selective write-back of records to the disk by MAINT.

A local field "W$W" of type integer is introduced. After each record is processed but before it is written back to the disk the local field W$W is examined by MAINT. If W$W has been set to "1" by the RMO statements in the course of execution of the RMO on the record, then and only then is the record written back to the disk. After the write-back MAINT will set W$W back to zero so W$W must be reset to "1" by the RMO for subsequent records to be written back. If W$W is not present at all as a local field in the RMO then **all** records are written back by MAINT.

10.5 Record Deletion: D$D

The RMO has the ability to tell MAINT to delete the current record from the file,[3] using the special local integer field **D$D.** The default for the delete option is no delete (D$D = 0). For a record to be deleted, the RMO must set D$D = 1. If D$D is 1 after the RMO executes on a record MAINT will delete that record, and then reset D$D to zero.

10.5.1 NOFLUSH: No Disk Write After Deletion

MAINT normally writes ("flushes") the file header, index blocks, and data blocks to the disk after every deletion.[4]

This protects file integrity should MAINT abort or the system crash while MAINT is running. However, flushing to the disk slows down deletion dramatically. A command line qualifier, "NOFLUSH", can be used to tell MAINT not to flush the file blocks to disk. NOFLUSH can speed up MAINTs which do a lot of (D$D) deletion considerably, but it must be used with caution. If MAINT does not complete normally, the file may be left in an unusable state. NOFLUSH should only be used in situations where improved performance is important, and where MAINT can be restarted with a current backup copy of the file.

If this option is used, NOFLUSH must appear on the MAINT command line and must be the last thing on the command line, as in the following example:

```
$ MAINT CLEANFILE/NOFLUSH
```

10.6 Quitting Before End of File: Q$Q

Normally MAINT processes the input file from beginning to end. However, there will be times when the RMO is only required to process the file up to a particular record and processing of subsequent records is not needed.

If the user introduces a local field Q$Q of type integer in the RMO then this field may be used to instruct MAINT to stop processing at a given record, close the file and terminate. That is, the RMO should contain statements which set Q$Q to "1" at the record where MAINT is to quit. MAINT examines Q$Q after each record, and when MAINT sees "1" in Q$Q, MAINT stops processing.

3. The file must be in sort order.
4. MAINT deletes records with the D$D facility, described in Section 10.5 “Record Deletion: D$D”.

10.7 Terminating a Command File: E$XIT

A MAINT may be included in a series of commands called a "command file" described in Chapter 14: "Command Files". A running command file may be terminated after the processing of a MAINT. If the user introduces a local field E$XIT of type integer in the RMO then this field may be used to instruct MAINT to terminate the command file. This is done by having the RMO set the E$XIT local field to "1". The command file will be terminated at the end of the step containing the MAINT.

10.8 Printing On-line Messages: P$P

The user may wish to print on-line messages from a running RMO. This is done by introducing a local field P$P of type "An". When an alphanumeric string is placed in P$P field, the alphanumeric string is printed on-line immediately, P$P is reset to blank, and the RMO continues with the next statement. The line of asterisks are suppressed when P$P is used.

10.8.1 Example Using PP, WW, and Q$Q

A procedure is to be written to find errors in a file of purchase orders. The file is keyed and sorted on purchase order number (PO#). The procedure is to find those purchase order records in a purchase order file where the expended amount (EXPEND) exceeds the encumbrance (ENCUMB). These purchase orders are to be marked with an 'X' in the ERROR field. The PO# for these purchase orders are to be printed on-line. This procedure is only to be applied to purchase orders up to PO "P0999" That is, MAINT may stop processing as soon as 'P0999' is reached.

The following RMO uses the WW, QQ and P$P local fields to accomplish these objectives.

```
FILE PO.MAS
LOCAL
W$W/I
Q$Q/I
P$P/A20
MSG/A16 'IS INCORRECT'
BLANK/A1 ' '
PROGRAM
IF PO# GE P0999 THEN Q$Q = 1 END
IF EXPEND GT ENCUMB THEN
  W$W = 1 ; ERROR = 'X' ;
  P$P = NCAT(P$P,PO#,BLANK,MSG) ; END
```

10.9 Internal Fields: TODAY, NOW, and TICKS

TODAY/DA or TODAY/DT, NOW/A8 or NOW/TM, and TICKS/I may be introduced as local fields in the RMO. MAINT will see that they are set to the current date (TODAY), the current time to the hour, minute, and second (NOW/A8),[5] and the hundredth of a second of the current time (NOW/TM or TICKS), before each record is read and the RMO is executed on that record.

10.10 Backspace Records: BACKSPACE

A running RMO can request that MAINT backspace any number of records. This is done by setting the local integer BACKSPACE field to the number of records to be backspaced. At the next call to the RMO the backspace will have been performed, and the BACKSPACE local field will have been reset to zero. Setting BACKSPACE to a negative value can be used to space forward a number of records.

10.11 Look Ahead: NX$fieldname

One can examine values of fields in the record following the current record. For each field that is to be examined declare a local field called "NX$fieldname". For example, to examine ACCT in the next record create NX$ACCT.

The "NX$fieldname" field will always be set to the value of "fieldname" from the record following. If the current record is the last record in the file then an integer field called "NX$EOF", if declared, will be set to "-1" and the "NX$fieldname" values will remain unchanged from their last setting.

5. If NOW is declared as field type TM, NOW will be set to the current time to the hundredth of a second. TICKS requires the presence of the NOW local field.

10.12 Writing Other Files: OUTFILE/OUTRECS

Data present in tables, arrays and local fields can be output after MAINT completes the processing of the input data file. This is done by setting local fields OUTFILE/An and OUTRECS/I to the output file name and output number of records respectively. After MAINT finishes processing the input file it will check the contents of these two fields. If OUTFILE is not blank then it will open the file named in OUTFILE, and **append** the number of records specified in OUTRECS into the named file. Local fields from the RMO are placed in fields of the **same name** in the output file. (If OUTRECS is greater than 1, then local fields being appended should either be arrays or tables.) After closing the first output file, MAINT will call the RMO again with OUTFILE set to blank. If the RMO sets OUTFILE and OUTRECS again then MAINT will append records to the newly named output file. This process will continue until the RMO leaves OUTFILE blank. In this way the RMO can append data produced by processing the input file into multiple output files.

10.12.1 OUTFILE Example

The use of OUTFILE should be considered an advanced technique of ADMINS. Usually the file created by MAINT could just as well be created using more conventional ADMINS commands and techniques, although possibly somewhat less efficiently. For example, a primary file is being processed by MAINT as part of a routine process. The writing of an output file(s) may be used to produce summary records. This may be slightly more efficient then using SORT to create the summary records. However, the SORT method will invariably be easier to program and maintain.

Some of the potential uses of OUTFILE are:

1. Re-write a TABLE file that has been updated by MAINT. This is done by writing the TABLE array at end of file.
2. Create a file control record containing number of records and totaled values for several fields which may be needed for audit purposes. This is done by totaling into and then writing local fields.
3. Create control values for separate groups of records in a file, e.g. batch controls, to be used for audit purposes. This is done by building and then writing a local array.
4. Create one or more transaction records from a single master record based on the content of the master record. For example, generate all the applicable payroll earnings records for a terminating employee based on rules governing payoff of accrued vacation time, sick leave, holiday, compensatory time, etc. Again this done by writing a local array.

The following example shows the output of a TABLE which has been modified by MAINT. The case illustrated assigns a unique batch number to a file and a unique vendor sequence number to each record in a file. When finished, we output the TABLE record replacing the previous copy of the one record NBR.TAB file so that the next time the MAINT is used it will use the next available batch and vendor sequence number. We show the DEF of the table file, the RMS, and the dialogue which runs the procedure.

```
* NBR.DEF
*
TAB 1
*
BATCH I KEY1   "Last Batch Number Used"
VENDOR D       "Last Vendor Sequence Number Used"
```

```
*  REFUND.RMS
*
FILE REFUND.MAS
TABLE NBR.TAB BATCH VENDOR
LOCAL
COUNT/I
SW/I
OUTRECS/I 1
OUTFILE/A16 'NBR.TAB'
PROGRAM
IF SW EQ 0 THEN SW = 1 ; BATCH = BATCH + 1 END
BCH = BATCH
COUNT = COUNT + 1 ; ITEM = COUNT
VENDOR = VENDOR + 1 ; VEN# = VENDOR
```

First, compile REFUND.RMS loading the current record from NBR.TAB into the TABLE array.

```
$ cmp refund
TABLE NBR.TAB: 1 ENTRIES LOADED INTO DATA BLOCK
REFUND.RMO WRITTEN. 4 PARAGRAPHS. OBJ-SIZE: 52  CONS-SIZE: 8
```

Second, change the name of NBR.TAB so we can create a new empty copy. (XNBR.TAB can be deleted after the procedure has been completed successfully.)

```
$ rename nbr.tab xnbr.tab
```

Next, create a new empty copy of NBR.TAB for the MAINT to use.

```
$ define
FILE NAME:nbr
NBR.TAB CREATED
```

Finally, run the MAINT writing the updated TABLE array to NBR.TAB. The updated NBR.TAB is ready for the next run of this process.

```
$ maint refund
TEST?  (Y OR N):n
OPERATING ON REFUND.MAS
OK TO CONTINUE?y
20:51:54.74
                                                          **********

1 RECORDS APPENDED INTO NBR.TAB
10 RECORDS PROCESSED   20:51:55.44
$
```

10.13 Rebuilding Indices After Batch Processing

MAINT allows you to rebuild indices, on normal termination, that may have been invalidated by batch commands. This rebuild of the indices uses AdmSort and requires the use of the command line switch **-SORT**.

The option is also available in MOVE: Section 3.2.6 "SORT: Rebuild indexes after records moved".

Chapter 11:PROD: File Linkage Relational Product

The PROD command that operates on whole files at a time, links records in two different files, and then performs an action that can alter one (or both) of the records from the two input files, and/or produce a combined record that is output to a third file.

11.1 Conceptual Description

The first of the two input files is called the **detail** file. The records from the detail file are read in sequence from the start of the detail file to the end of the detail file. The second of the two input files is called the **lookup** file. The records in the lookup file are directly accessed via their key values.[1] That is, they are looked up by their key values using fields from the record in the detail file to form the identifying key values.

After each detail record is read into memory, an attempt is made to find and read into memory the lookup record identified by the "link" fields in the detail record. When a record from the detail file and a record from the lookup file are linked together in memory there are several possible actions that can then be performed by PROD. (Any or all of these actions can be performed.)

1. Data can be moved from the lookup file record to the detail file record, and the record from the detail file record containing this new (or altered) data can then be written back onto the disk into the detail file.
2. Data can be moved from the detail file record to the lookup file record, and the record from the lookup file record containing this new (or altered) data can then be written back onto the disk into the lookup file.
3. Data from both the lookup file record and the detail file record can be combined into a new record that is appended to an output file.

1. The lookup file must be in sort. Records not in sort order cannot be directly accessed.

In any of these cases a record maintenance procedure can operate on the data in memory **when a link is achieved** between the detail and lookup files, and before the records are written back to the disk. The RMO is written to run on either the detail, lookup or output files. The records in the file for which the RMO is written are called the RMO records. If there is a record maintenance procedure active under PROD, then that record maintenance procedure can have "local" fields to receive data from the non-RMO record without necessarily filing that data in actual fields of the RMO record. The decision whether or not to file data from the non-RMO record received in a local field of the RMO record can be made by the record maintenance procedure itself when it is executing. (Data in local fields in record maintenance procedures exist only in memory and are never written out to the disk. Local fields are not initialized from record to record, i.e. they can be used to hold values as PROD proceeds in its record-by-record processing.)

The lookup file can contain multiple records with the same key value(s). In this case **each matching record is processed against the detail record** that generated the link fields.

The detail file need **not** be in any particular sort order. If, however, the detail file is in order on the link fields, i.e. the fields from the detail file that are used to form the key into the lookup file then PROD is optimized to take advantage of this and will run much faster.

11.2 Detail File

The dialogue that ensues when PROD is run is as follows:

```
$ prod
  Detail file......:det-file-spec [w]
  Link fields....:lf1 lf2 lf3 etc.
  Transfer fields:tf1 tf2 tf3 etc.
```

The first part of the dialogue concerns the specification of the **DETAIL** file. First PROD asks for the name of the detail file, which the user types in response to the "Detail file:" prompt.

If a write-back is designated by placing the "W" after the detail file name then after data from the lookup file record is moved into the detail file record, the detail file record will be re-written to the disk.

When PROD is executing, it displays a line of asterisks to show its progress through the detail file. A "NO *" typed to the "Detail file:" prompt will suppress the line of asterisks and PROD will prompt for the detail file again.[2]

PROD can be instructed to skip n records in the detail file before starting the detail/look-up linkage process. This is done by typing "skip n" to the "Detail file:" prompt. PROD displays "n RECORDS WILL BE SKIPPED", and then gives another "Detail file:" prompt.

2. If the character "*" (asterisk) is included in the string assigned to the logical name OPTION, the printing of the line of asterisks to show progress through a file is suppressed in all ADMINS "batch" commands. See Appendix A: “Options”.

After the detail file specification is entered, PROD prompts for the "Link fields:". These are names of fields in the detail file that taken together in the order they are typed, constitute a key value that can be searched for in the lookup file. (If the user types the letter "F" or "?" for fields in response to any "Link fields:" or "Transfer fields:" prompt, then PROD will print a list of all fields in the detail file, and re-prompt for the fields). The link fields need not have the same name as the key fields in the lookup file. However, they must match the lookup key fields in type.

Then PROD prompts for the transfer fields. These are the names for the fields in the detail file that will provide/receive data to/from the lookup record when the detail file record is linked in memory with the lookup file record. PROD can transfer up to 200 fields. All transfer fields do not have to be entered on the same line. If PROD receives a line ending with a "colon" that is preceded with a blank, PROD will prompt for "Transfer fields" again. This continues until PROD receives a line that does not end with a "colon". For example:

```
Transfer fields:tf1 tf2 :
Transfer fields:tf3
Lookup file......:
```

A through notation ("-") in the transfer fields for both detail and lookup allows the transfer of a large number of fields with a simple instruction (i.e. "1FLD - 7FLD" rather than "1FLD 2FLD 3FLD etc....7FLD"). However, these fields must be in the same sequence and have the same field types in both files.

If neither file is being accessed via a record maintenance procedure then data is moved from the non-write-back record into the write-back record. If there is a record maintenance procedure involved, the field movement and write-back are described in Section 11.7 “Use of Record Maintenance Procedures”.

11.2.1 PROD with Key Range: PROD/KEY

If the qualifier "KEY" (for "key range") appears on the command line PROD will prompt for a key range to be processed. The key range specifies the records in the Detail file which are to be processed.[3]

After the name of the Detail file has been provided PROD will prompt for the "Start of key range", followed by a prompt for the "End of key range". If the input file has multiple keys, enter all or some of the key values, separated by a blank. Null values are used for minor keys not entered.

If the user responds to the "Start of key range" prompt with a question mark (?), PROD will display a list of the key fields and their field types, and then re-prompt for the starting value.

3. Use of the key range requires that the file be in sort. PROD with key range exits as soon as it encounters a Detail file record with a key that exceeds the high key value of the specified range. Any subsequent (out of sort) records with key values in the specified range will not be processed.

PROD will accept logical names for the key range prompts. If the response either prompt begins with the letters "L$" then PROD will attempt to translate the response as a logical name. If such a logical name exists, PROD will use the string assigned to it as the (low or high) key value, otherwise PROD will use the response directly as it usually does.

When the end of the key range value has been entered, PROD prompts for the Link fields.

11.2.2 Wildcard and Copy Syntax for Transfer Fields

PROD has a '*' wildcard notation for transfer fields. The '*' transfers all fields with the same names between the detail and the lookup file, or between the lookup and the output file. The fields can be in any order in the DEFs, or can be local RMO fields. The '*' should appear on the transfer fields line for each of the two files between which the wildcard transfer is to be made. The '*' can be combined with explicit transfer fields. If a field which has the same name in both files is explicitly named as a transfer field, the explicit transfer overrides the wildcard transfer. To protect file integrity, lookup key fields are skipped by wildcard transfers between detail and lookup (if desired, explicit transfer fields can be used to transfer lookup file keys). In all other respects, wildcarded transfer fields operate exactly as though you had specified the transfer fields explicitly.

The previous set of transfer fields can be copied by responding to the LOOKUP or OUTPUT "Transfer fields:" prompt with an equals sign (=), as in the following example:

```
$ prod
Detail file......:new.mas
  Link fields....:item
  Transfer fields:sdesc unit unitpr shipwt category
Lookup file......:catalog.mas wi
  Link fields....:item
  Transfer fields:=
```

In the above example, the "=" response to the LOOKUP file transfer fields prompt is equivalent to typing in the same field names given to the previous (DETAIL) transfer fields prompt.

11.3 Lookup File

The next part of the PROD dialogue concerns the lookup file and proceeds as follows.

```
Lookup file......:lkup-file-spec [w][i]
  Link fields....:lf1 lf2 lf3 etc.
  Transfer fields:tf1 tf2 tf3 etc.
```

This dialogue provides the same information for the lookup file as the previous dialogue provided for the detail file. The "colon" continuation method for entering the transfer fields applies as well.

However, more checking is performed here based on what is already known from the detail file specification. One, the LINK FIELDS must be key fields in the lookup file, and they must match in type, although not in name, with the LINK FIELDS provided in the detail file part of the dialogue. Two, the transfer fields must also match in type, although not in name, with the transfer fields provided for the detail file.

The write-back designation, "W", is possible even if the detail record is being written back as well. This is because when record maintenance procedures are used, write-back can be requested both for records in the detail and lookup files.

Records from the detail file may be "inserted" into the lookup file if they are not found in the matching process. This is indicated with the optional "I" following the lookup file specification and is described fully in Section 11.6 “Inserting In The Lookup File”.

11.4 Output File

The final portion of the dialogue concerns the output file. (If there is not to be any output file then the user presses carriage return after the "Output file:" prompt.)

```
Output file......:out-file-spec
  Transfer fields:tf1 tf2 tf3 etc.
```

If there is an output file then every time a detail record links in memory to a lookup record the fields listed under TRANSFER FIELDS for the lookup record are moved into the fields listed under TRANSFER FIELDS for the output file.

Also all fields of the same name are moved from the detail file to the output file. Therefore if there is an output file, the user does not type any transfer fields for the detail file; the user simply presses carriage return to the prompt.

The output file record then is appended to the output file.

It is possible to write back detail and/or lookup files while creating an output file. In this case the RMO would operate on the output file.

The output file does not have to be empty when a PROD that produces output records is started.

11.5 PROD Examples

In this example, we wish to add vendor name and address into a purchase order file which is in no particular order. The purchase order record (detail file) contains the vendor number (VEND#) which is also the key field in a vendor table file (lookup file).

```
$ prod
Detail file......:po.mas w
  Link fields....:vend#
  Transfer fields:vendor addr
Lookup file......:vendor.tab
  Link fields....:f
VEND#/K1 NAME ADDRESS
  Link fields....:vend#
  Transfer fields:name address
Output file......:cr
17:54:31.04
************************************************************
17:55:28.53
188 records updated in PO.MAS
$
```

Note the TRANSFER FIELDS have different names in the detail and lookup records. Also note the use of the "F" entry to display the field names from VENDOR.TAB.

In the following example, we wish to add records to a file used for printing "reminder" notices to customers with overdue accounts. The batch of overdue invoices (in no particular order) is the detail file, INV.MAS. It contains the invoice number (INVOICE), the customer ID number (CUSTID), the amount due (AMT) and due date (DUEDATE). The lookup file is the customer information file, CUSTOMER.MAS, which contains name and address information. CUSTOMER.MAS, has CUSTID as its key, and is in sort. When PROD links the record from CUSTOMER.MAS identified in the CUSTID field of INV.MAS, it will add a record to REMINDER.MAS, the output file. The reminder records will contain the customer ID, the invoice number, the amount and due date, as well as the customer's name and address.

```
$ prod
Detail file......:inv.mas
  Link fields....:custid
  Transfer fields:cr  "note, no transfer fields are entered"
Lookup file......:customer.mas
  Link fields....:f
CUSTID/K1 NAME 1ADDR 2ADDR CITY STATE ZIP
  Link fields....:custid
  Transfer fields:NAME 1ADDR 2ADDR CITY STATE ZIP
Output file......:reminder.mas
  Transfer fields:f
CUSTID/K1 INVOICE NAME 1ADDR 2ADDR CITY STATE ZIP AMT DUEDATE
  Transfer fields:name 1addr 2addr city state zip
15:31:06.15

************************************************************
244 records updated in REMINDER.MAS
15:32:08.96
$
```

The lookup transfer fields are moved into the output transfer fields (in this example they have the same names). Note that no detail file transfer fields are entered when there is an output file. The fields in the detail file that have the same name in the output file (INVOICE, AMT, and DUEDATE) are moved from DETAIL to OUTPUT.

11.6 Inserting In The Lookup File

PROD can be instructed to insert records into the lookup file in those cases where the link from detail to lookup fails because the sought-after record is not present in the lookup file. This feature is invoked by placing "WI" after the lookup file specification. "W" for write-back on the lookup and "I" for insert if the record is not found. If there is an RMO on the lookup file, the RMO is executed on the inserted record.

For example, if we had the following two files:

```
    DET.MAS              LKUP.MAS
ACCT       AMT       ACCT       AMT
  1        110         1         25
  2        250         5          0
  5        860        10        500
  7        900        15          0
 10       1145
```

We could run the PROD as follows:

```
$ prod
Detail file......:det.mas
  Link fields....:acct
  Transfer fields:amt
Lookup file......: lkup.mas wi
Operating on LKUP.MAS
  Link fields....:acct
  Transfer fields:amt
Output file......:cr
13:12:36:97
                                                              *****
5 records updated in LKUP.MAS
13:12:37.13
$
```

After PROD has been run the contents of LKUP.MAS would be as follows.

```
    LKUP.MAS
ACCT       AMT
  1        110
  2        250
  5        860
  7        900
 10       1145
 15          0
```

PROD can also be instructed to insert **unconditionally** in the lookup file, i.e. insert a record in the lookup file for every record in the detail file whether or not a record with that particular key value is already present in the lookup. This is accomplished by placing "WA" following the lookup file specification. "W" for write-back on the lookup and "A" for always insert the record. As with "WI", if there is an RMO on the lookup file, the RMO is executed on the inserted records.

If "WA" had been specified in the above example the contents of the resultant lookup file would then be as follows:

```
    LKUP.MAS
ACCT       AMT
  1        110
  1         25
  2        250
  5          0
  5        860
  7        900
 10        500
 10       1145
 15          0
```

11.6.1 PROD/NOFLUSH: No Disk Writeback with Insert/Delete

PROD normally writes ("flushes") the lookup file header, index blocks, and data blocks to the disk after every insertion or deletion. This protects the integrity of the lookup file if the PROD should abort or the system crash while the PROD is running. However, flushing to the disk slows down insertion and deletion dramatically. There is a command line option, /NOFLUSH, that tells PROD not to flush lookup file blocks to disk. NOFLUSH can speed up PRODs which insert or delete by about a factor of three; but it must be used with caution. If the PROD does not complete normally, the lookup file may be left in an unusable state. /NOFLUSH should only be used in situations where improved performance is important, and where the PROD can be restarted with a current backup copy of the lookup file.

11.7 Use of Record Maintenance Procedures

Either the detail, lookup or output file name that is provided in the dialogue can be a record maintenance procedure object file name, where the record maintenance procedure has been set up to operate on the (detail, lookup or output) file. However only one RMO can operate in a PROD run. In this case the record maintenance procedure is executed after the records are linked in memory and before they are written back to the disk. The step by step process in which record maintenance is involved is as follows:

1. The records are linked in memory.
2. Data is moved from the non-RMO record to the RMO record.
3. The RMO is executed.
4. If the non-RMO record is to be re-written to the disk, data is moved from the RMO record to the non-RMO record.
5. The records designated for "write-back" are written to the disk.

11.7.1 Test Mode in PROD

PROD provides a Test Mode for testing RMO steps that operates in exactly the same way that Test Mode operates in MAINT (see Section 10.2.2 “Test Mode Operation”).[4]

Test Mode in PROD is requested by placing the qualifier "TEST" on the command line:

```
$ PROD/TEST
Detail file......:
```

4. As in MAINT, PROD Test Mode does not actually update any of the files.

11.8 PROD Example Using An RMO

For example, the following record maintenance procedure posts payments to a general ledger. Also, if the identification (IDENT) of the payment is a purchase order number in the range of P0000 to P9999 then the encumbered field in the general ledger is liquidated for the value of the payment. Also, the VERIFY field on the payment record is set with 'P' for posted. Therefore, all payment records that don't have 'P' in the VERIFY field after PROD has been run were not able to link to a general ledger account, i.e. the payment record had an invalid account number.

```
*          POSTPA.RMS
*
FILE GL.MAS
LOCAL
IDENT/XA9999
PAY/D2
VERIFY/A1
PROGRAM
PAID = PAID + PAY
VERIFY = 'P'
IF IDENT BET P0000 AND P9999 THEN
     ENCUMB = ENCUMB - PAY END
```

```
$ prod
Detail file......:paym.mas w
  Link fields....:f
IDENT/K1 FUND DEPT SERV OBJ PAY VERIFY
  Link fields....:fund dept serv obj
  Transfer fields:ident pay verify
Lookup file......:postpa.rmo w
Operating on GL.MAS
  Link fields....:f
FUND/K1 DEPT/K2 SERV/K3 OBJ/K4 APPR ENCUMB PAID IDENT
PAY VERIFY
  Link fields....:fund dept serv obj
  Transfer fields: ident pay verify
Output file......:cr
12:25:56.03
************************************************************
179 records updated in PAYM.MAS
179 records updated in POSTPA.RMO
12:26:48.44
$
```

11.9 Internal Fields: TODAY, NOW and TICKS

The special internal fields TODAY/DA or TODAY/DT, NOW/A8 or NOW/TM, and TICKS/I may be introduced as local fields in the RMO running with PROD. If they are present, PROD loads them with the current date (TODAY), the current time to the hour, minute, and second (NOW/A8),[5] and the hundredth of a second of the current time (NOW/TM or TICKS), before each record is read and the RMO is executed on that record.

The following example is used to post payments from the detail file to the lookup file and keep a log of all payments posted in the output file. The output file is keyed on DATE, TIME and TK.

```
*     DET.DEF
*
MAS 1000
TNO I KEY1   "transaction #"
ACCT# XA999999
AMT D

*     LKUP.DEF
*
MAS 500
ACCT# XA999999 KEY1
BAL D

*     OUT.DEF
*
MAS 1000
DATE DA KEY1
TIME A8 KEY2
TK I KEY3
AMT D
TNO I
ACCT# XA999999

*     OUT.RMS
*
FILE OUT.MAS
LOCAL
TODAY/DA
NOW/A8
TICKS/I
BAL/D
PROGRAM
DATE = TODAY ; TIME = NOW ; TK = TICKS
BAL = BAL - AMT
```

5. If NOW is declared as field type TM, NOW will be set to the current time to the hundredth of a second. TICKS requires the presence of the NOW local field.

The PROD dialogue would be as follows:

```
$ prod
Detail file......:det.mas
  Link fields....:acct#
  Transfer fields:cr
Lookup file......:lkup.mas w
Operating on LKUP.MAS
  Link fields....:acct#
  Transfer fields:bal
Output file......:out.rmo
  Transfer fields:bal
12:22:21.34
************************************************************
388 records updated in LKUP.MAS
388 records updated in OUT.RMO
12.23.57.65
$
```

11.10 NOMATCH qualifier: Functionality without LOOKUP link

The "NOMATCH" command line qualifier allows PROD to perform most of its functions even if no link is made to the lookup file. With NOMATCH in effect, PROD acts normally if there is a link to a record in the LOOKUP file. However, if there is no link and NOMATCH in effect, the RMO executes, the detail file can be written to, and records can be appended to the output file (if any).

NOMATCH has a facility for identifying records for which NOMATCH processing was in effect. If the special integer field PROD$LINK is a local field in the RMO, it is set to zero when there is no link, and is non-zero when a link is made. PROD$LINK can be used, for example, to identify and mark detail records which are not in the lookup file

The special NOMATCH RMO local integer field I$I enables the RMO to control insertion into the lookup file. To use I$I, NOMATCH must be in effect, and writeback ("W", not "WI" or "WA") must be specified for the lookup file. The special field PROD$LINK, explained above, should be in the RMO. When PROD$LINK is zero, the record does not exist in the lookup file, and I$I can be set to 1 to insert the record in the lookup file. When I$I is set to 1, the record is inserted. PROD then automatically sets I$I to zero and sets PROD$LINK to a non-zero value, and the RMO is called again. At this second call, transfer fields can be set and normal updating can be performed.

11.11 Controlling Writeback and Output: W$W

There are situations where appending to the output file or writing back to the detail or lookup file is a conditional action depending on the values in the detail and/or lookup files. This conditional control functionality is available in PROD through the use of the special local field, W$W. To extend the previous example in Section 11.9 "Internal Fields: TODAY, NOW and TICKS" to illustrate this facility say we did not wish to create output records if the balance in the lookup file was zero. We use the W$W/I field to instruct PROD whether or not to append to the output file. That is, if W$W is present in the output file RMO, then it must be set to 1 at each call to the RMO in order for PROD to create an output record.

Therefore we would alter OUT.RMS as follows:

```
        *     OUT.RMS
        *
        FILE OUT.MAS
        LOCAL
        TODAY/DA
        NOW/A8
        TICKS/I
        BAL/D
    --> W$W/I
        PROGRAM
        DATE = TODAY ; TIME = NOW ; TK = TICKS
    --> IF BAL NE 0 THEN W$W = 1 ; BAL = BAL - AMT ELSE W$W = 0 END
```

The following table lists the only valid settings for W$W, along with their corresponding control functions.

For PROD with OUTPUT file:

set W$W to:	Functionality
0	No append to OUTPUT are performed, (DETAIL and LOOKUP writebacks ALWAYS occur in PROD with OUTPUT!)
1	Append record to OUTPUT

For PROD without OUTPUT file:

set W$W to:	Functionality
0	No writebacks are performed
2	Writeback to DETAIL only
4	Writeback to LOOKUP only
6	Writeback to both DETAIL and LOOKUP

11.12 Controlling Lookup File Insertion: DI$DI

There is facility for controlling the insert to the lookup file. The detail record can contain an **actual** field of DI$DI/I. If this field is set to zero in a particular detail record, then that detail will never create an insert into the lookup file.

11.13 Record Deletion: D$D

The RMO has the ability to delete records from the lookup file through the use of a local integer field named D$D. The default for the delete option is **no** delete (D$D = 0). For a record to be deleted, the D$D field must be set to one. That is, after the RMO executes on the RMO record in the lookup file, the RMO can request that PROD delete that record from the lookup file by setting D$D to "1". PROD will delete the record and reset D$D back to zero.

11.14 Terminating a Command File: E$XIT

A PROD may be included in a series of commands called a "command file", which is described in Chapter 14: “Command Files”. A running command file may be terminated **after** the processing of a PROD. If the user introduces a local field E$XIT of type integer in the RMO then this field may be used to instruct PROD to terminate the command file. This is done by having the RMO set the E$XIT local field to "1". **The command file will be terminated at the end of the step containing the PROD.**

11.15 Quitting Before End of File: Q$Q

Normally PROD processes the detail file from beginning to end. However there will be times when PROD is only required to process the file up to a particular record and subsequent record processing is not needed.

If the user introduces a local field Q$Q of type integer in an RMO with the PROD, then this field may be used to instruct PROD to stop processing at a given record, close the file and terminate. That is, the RMO should contain statements which set Q$Q to "1" at the record where PROD is to quit. PROD examines the Q$Q after each record, and when Q$Q is "1", PROD stops processing.

11.16 Itemization and De-Itemization

PROD can also be used to place fields from **several records** of the same key value into **one large record** with repeating fields, and conversely place fields from **one large record** with repeating fields into **several smaller records.** We call this "de-itemizing" (several records into one record) and "itemizing" (one record into several records). The several itemized records produced from the same de-itemized record always contain the same key.

Let us consider a de-itemizing example. Individual payment records contain the vendor number, the invoice number and the payment amount. We wish to print checks where a vendor receives one check for all payments, and the check stub shows the individual amounts and invoice numbers. (Note the request for field names is optional and there is no lookup file.)

```
$ prod
Detail file......:paym.mas
  Link fields....:f
VEND#/K1 INVOICE AMT
  Link fields....:vend#
  Transfer fields:invoice amt
Lookup file......:cr
Assuming ITEMIZE/DE-ITEMIZE
Output file......:checks.mas
  Transfer fields:f
VEND#/K1 CK# VENDOR ADDR CITYST INV1 AMT1 INV2 AMT2 INV3
AMT3 INV4 AMT4 INV5 AMT5
  Transfer fields:inv1 - amt5
```

When using PROD to itemize/de-itemize there is no lookup file, and the link fields of the detail file are the keys in the detail file. The through notation (see the "-" in the output file transfer fields in the above example) is used to designate the repeating fields. When PROD is itemizing the repeating fields are designated by the through notation in the detail file, and when PROD is de-itemizing then the repeating fields are designated by the through notation in the output file. Repeating fields can only be designated in one or the other and through notation may only be used once. The repeating fields are always repeating consecutive sequences of fields that match the itemized fields in type and length.

PROD, when itemizing, will stop generating itemized records whenever a particular group of fields with null values in the transfer fields is reached. Therefore, if a file contains repeating groups of fields and is to be subsequently itemized, be sure that all the non-null groups of values are placed consecutively in the repeating groups.

When de-itemizing, the DETAIL file should be in sort because PROD will create a new OUTPUT record whenever the key changes in the DETAIL file. Itemized records with the same key value that are not adjacent will not be combined in the de-itemized output record.

If PROD runs out of fields in the de-itemized record, i.e. there aren't enough repeating groups of fields for all the repeating records in the detail, then PROD will generate an additional de-itemized record with the same key value(s) so as to include all the values from the detail (itemized) file.

Also, as in the general use of PROD, other fields of the same name are moved from the detail record to the output record.

Let us reverse the above example to illustrate itemization. We have a record that contains the vendor number, name, address and check number and five pairs of invoice numbers and amounts. We wish to create a record for **each** invoice/amount pair keyed on vendor number and also containing check number. We proceed as follows:

```
$ prod
Detail file......:checks.mas
  Link fields....:f
VEND#/K1 VENDOR ADDR CIYTST CK# INV1 AMT1 INV2 AMT2 INV3
AMT3 INV4 AMT4 INV5 AMT5
  Link fields....:vend#
  Transfer fields:inv1 - amt5
Lookup file......:cr
Assuming ITEMIZE/DE-ITEMIZE
Output file......:detail.mas
  Transfer fields: invoice amt
...
```

This would produce detail records per each invoice/amount pair keyed on vendor number with the check number (CK#) included. Note, CK# is moved implicitly from CHECKS.MAS to DETAIL.MAS. That is, as we stated above, fields of the same name are moved from the detail to the output file. Note also, that although each record in CHECKS.MAS can contain **up to** five invoice/amount pairs, records are created in DETAIL.MAS for **actual** invoice/amount pairs, i.e. when PROD comes to an empty (i.e. zero) INVn/AMTn pair in a record of CHECKS.MAS, PROD goes on to the next record in CHECKS.MAS.

11.17 Multiple Lookup Files

PROD can use more than one lookup file in relation to a detail file. This is done by keeping the name of the lookup file to which a detail record should be linked in a field of the detail record, and instructing PROD to get the lookup file name indirectly through this detail field as it is processing the detail records. Consider the following simple example:

```
F.DEF           F1.DEF          F2.DEF          F3.DEF

MAS 100         MAS 100         MAS 100         MAS 100
ID XA99 KEY1    N I KEY1        N I KEY1        N I KEY1
N I             SP A10          SP A10          SP A10
SP A10
FILE A10
```

These files are defined and the following data is placed in each:

```
F.MAS                   F1.MAS      F2.MAS      F3.MAS

ID  N SP    FILE        N SP        N SP        N SP
A01 1 ONE   F1.MAS      1           1           1
A02 2 TWO   F2.MAS      2           2           2
A03 3 THREE F3.MAS      3           3           3
```

PROD can be used to post the spellings to the three files as follows: Note the use of the parentheses in the entry to the "Lookup file" prompt. The parentheses indicate that the entry is a field in the detail file. (The fact that the FILE field is an alphanumeric field indicates to PROD that FILE contains the name of the lookup file for each record.)

```
$ prod
Detail file......:f.mas
  Link fields....:n
  Transfer fields:sp
Lookup file......:(file) w
  Transfer fields:sp
Output file......:cr
12:24:28.06                                                          ***

3 records updated in (FILE)
12:34:30.20
$
```

When we now inspect the contents of the 3 lookup files we find the following:

```
F1.MAS          F2.MAS          F3.MAS

N SP            N SP            N SP
1 ONE           1               1
2               2 TWO           2
3               3               3 THREE
```

A Record Maintenance Procedure (RMO) is not allowed on any of the multiple lookup files. Also, PROD will skip the record if the field designated to contain the lookup file name is blank.

11.17.1 Keeping Multiple Lookup Files Open

There is an alternate way to set up PROD to use multiple lookup files that is more efficient. However this alternate way could only be used with a limited number of lookup files.

When the lookup file is specified by name in a field of the detail file, only one lookup file is kept open at a time. That is, each time the lookup file is changed, the current one is closed and the next one is opened. This results in somewhat slower processing than if all the lookup files were kept open throughout the PROD run, but on the other hand there is no limitation placed on the number of lookup files referenced in the detail file.

The alternate method opens all the lookup files at once. The PROD will run faster because there will be no file closings and openings during the run.

In this alternate method the field in the detail record that specifies the lookup file is an integer field rather than an alphanumeric field. This field specifies whether a particular detail record links to the first, second, third, etc. lookup file. PROD will prompt for the names of the lookup files when it gets an integer "indirect file name" field.

In the previous example if F.DEF contained an integer field FILNO the PROD dialogue would go as follows:

```
$ prod
Detail file......:f.mas
  Link fields....:n
  Transfer fields:sp
Lookup file......:(filno) w
File name 1:f1.mas
File name 2:f2.mas
File name 3:f3.mas
File name 4:cr
  Transfer fields:sp
Output file......:cr
12:24:28.06
                                                                        ***
3 records updated in (FILNO)
12:34:30.20
$
```

A Record Maintenance Procedure (RMO) is not allowed on any of the multiple lookup files. Also, PROD will skip the record if the field designated to contain the lookup file number is zero.

11.18 LOOKUP Without an Exact Match

The link to the LOOKUP file normally either finds an exact match in the link file or, unless NOMATCH is in effect, goes on to the next record in the DETAIL file.

Four alternative linkage operations are available in situations when an exact match may not be found but when an actual link is desired. These operations compare the link key values to the key values in the LOOKUP file and link to the next higher or lower record in the LOOKUP file, when there is no exact match, or even if there is an exact match.

1. **LINKGT** - Link Greater Than: Links to the next higher (key value) record in the LOOKUP file even if there is an exact match. If there is none higher, PROD goes on to the next DETAIL file record. If multiple records in the LOOKUP file have the next higher key value, PROD links to each of these records.
2. **LINKGE** - Link Greater than or Equal to: Links to an all records in the LOOKUP file with an exact match, or if one is not found, links to all records with the next higher key value, as described above for LINKGT.
3. **LINKLT** - Link Less Than: Links to the next lower (key value) record in the LOOKUP file even if there is an exact match. If there is none lower, PROD goes on to the next DETAIL file record. If multiple records in the LOOKUP file have the next lower key value, PROD links to each of these records.
4. **LINKLE** - Link Less than or Equal to: Links to an all records in the LOOKUP file with an exact match, or if one is not found, links to all records with the next lower key value, as described above for LINKGT.

These alternative linking methods are activated using one of the four command line qualifiers, "LINKGT", "LINKGE", "LINKLT", or "LINKLE", as follows:

```
$ prod -linkle
Detail file......:det.mas
  Link fields....:acct
  Transfer fields:amt
Lookup file......: lkup.mas w
Operating on LKUP.MAS
  Link fields....:acct
  Transfer fields:amt
Output file......:cr
```

Chapter 12:ANALYZER: Generalized Data Analysis

The ANALYZER is a generalized data analysis tool. The ANALYZER may be used to examine a broad range of data from personnel or budgeting files to attitudinal surveys, from clinical records to inventory or accounting transaction files.

The ANALYZER may be used as an on-line ad hoc reporting tool for data retrieval, data checkout and report design, or as a tool to perform substantial analysis for policy and decision making purposes.

The ANALYZER consists of logic, calculation and aggregation operations that are used in a interactive step-wise fashion to examine, reorganize and reclassify data in ADMINS files according to some analytic purpose (the "analysis plan"). As well as providing the computational and manipulation facilities required for exploratory analysis, the ANALYZER is designed to act as a "professional assistant" for the end user data analyst.

The ANALYZER has a complete on-line HELP facility, which explains both the functions and the syntax of all the ANALYZER commands.

The ANALYZER remembers the steps that are used to manipulate data and can display the user's derived names and construction steps at any time. The ANALYZER keeps a log of all the work done in an analysis session, which allows the user to pick up from where the work was left, eliminating the concern of losing unsaved work. The step by step instruction log also allows the user to edit changes to an interactively developed analysis procedure and then re-run the whole procedure as a command file.

Usually after a brief training period, the data analyst begins to use the ANALYZER on his/her own data to accomplish some analytical or reporting objective.

12.1 Method of Operation

The ANALYZER operates on the "logical file" which includes the physical (ADMINS) data file, linked fields obtained by linking from the physical file to other physical files, and created fields created by calculation or logical operations involving fields in these physical files. To begin using the ANALYZER, one builds **sets** with the **SELECT** command to subset the items in the data file, based on values in the items, according to the hypotheses or questions being examined. A set is a list of items from the logical master data file. (For example, employees in a personnel file.) Once the first level[1] **sets** are built using codes and values in the data fields (e.g. a set of employees with five years or more of service) the set operations **INTERSECT, UNION, COMPLEMENT** and **EXCLUSIVE-OR** are used to combine and recombine the sets of items, according to one's analytical purpose. (For example, intersect the set of employees with over five years of service with the set of female employees.)

The **MARGINALS** instruction is used to have the ANALYZER automatically build first level sets either for coded data or numerical values. For numerical values the **MARGINALS** instruction builds interval sets, e.g. quartiles based on salary.

At any point in the process a detailed or summary table of the data in the sets can be requested in a variety of formats. These tables can include totaling operations, (some of which are automatic), percentages, and statistical indicators. After examining a table, the user can then build higher level sets based on what the table showed about the data.

If, during an analysis session, the user realizes that another related file is necessary for the analysis, the user simply **links** that file into the logical master file being analyzed and carries on with the analytical operations (e.g. linking in a salary history file).

Below some of the major concepts used in the ANALYZER are highlighted.

The "logical master file" includes all the fields from the physical ADMINS data file, linked fields, and created fields.

The ANALYZER starts with a physical ADMINS data file. PERSONEL.MAS contains personnel records.

```
AN> FILE PERSONEL.MAS
```

Fields can be linked from other ADMINS data files. The department name field, DEPTNAME, is linked from another file, DEPT.TAB.

```
AN> LINK DEPTNAME FROM DEPT.TAB KEY IS DEPT
```

Other fields can be created by defining a calculation or logical operation. The field YEARS contains the years of service.

```
AN> CREATE YEARS/I 85 - HIREYR
```

1. First level sets are expressed directly in terms of values in data fields, and are created using the SELECT or MARGINALS command. Second level sets are expressed both in terms of other second level sets and first level sets.

A set is a logical combination of characteristics in the data. First level sets are created using the SELECT or MARGINAL commands, to make the basic data characteristics accessible to the higher level set building commands of the ANALYZER.

A set is a named list of items from the logical master file. Employees in department 2020, are in the accounting dept. This set is named DEPT.ACCT.

```
AN> SELECT DEPT EQ 2020 S:DEPT.ACCT
```

A set can be based on the value of a created field, e.g. the field YEARS defined in a create above. Employees with 5 or more years of service are in the set named YRS:5+.

```
AN> SELECT YEARS GE 5 S:YRS:5+
```

Sets can be built automatically for coded or numerical values. A set of the female employees and a set of the male employees are automatically built (i.e. sets to the possible codes in the SEX field).

```
AN> MARG SEX
```

Sets can be built automatically for intervals of values. Sets are automatically built to 4 salary ranges, each with (approximately) the same number of people. That is, quartiles of salary.

```
AN> MARG SALARY/I4
```

Higher level sets are built using the set operations INTERSECT, UNION, COMPLEMENT, and EXCLUSIVE-OR, allowing the user to construct new analytic combinations of characteristics from the data.

INTERSECT builds a set of items present in both sets. A new set of female employees with at least 5 years of service is built by intersecting the set of employees with 5 or more years of service with the set of female employees

```
AN> INTERSECT YRS:5+ SEX.F = F.5+
```

UNION builds a new set of items present in either set. We create a set of employees with at least 5 years of service or in the 4th (the highest) salary range.

```
AN> UNION YRS:5+ SALARY.4/4 = HISAL.5YRS
```

COMPLEMENT builds a new set of items not in the first set. We create a set of employees with less than 5 years of service.

```
AN> COMPLEMENT YRS:5+ = YRS:UNDER5
```

The TABLE command is the tool for displaying the data and for summarizing the data via existing sets in preparation for further set construction.

Print a listing of the first and last names, addresses and salaries for all the employees in the personnel file, sorted by last name.

```
AN> TABLE FNAM LNAM/S1 ADDR CITY STATE SALARY
```

Print a detailed list of name, addresses, salaries and average age for employees ordered by length of employment. The employees are grouped into sets based on length of employment under 2 years, 2 to 5 years, or 5 years or more.

```
AN> TABLE YRS:<2 YRS:2-5 YRS:5+
VALUES: LNAM/D ADDR/D SALARY/D AGE/AV YEARS/S1
```

Print a report comparing the average salaries for women and men in the accounting department.

```
AN> TABLE
ACROSS THE TOP: DEPT.ACCT
DOWN THE SIDE:  SEX.F SEX.M
VALUES: SALARY/AV
```

The ANALYZER works as the user's assistant in managing the interactive analysis session.

SHOW can display the fields, sets, option settings, and the log of the analysis steps.

```
AN> SHOW SETS
```

EXAMINE gives a detailed breakdown of a set.

```
AN> EXAMINE HISAL.5YRS
```

The WRITE command packages the analysis into a text editable command file for future use.

```
AN> WRITE ANALYSIS
```

12.1.1 Reporting and Outputting Data Files

The TABLE command is the analytical reporting facility in the ANALYZER. Reports can be printed on the screen or in hardcopy. There are several report formats available within the ANALYZER. The reports can be sorted on any fields before printing. In addition, a variety of summary operations can be requested (e.g. average, maximum, percentages, totals etc.) The table layouts provided by the ANALYZER include:

1. Detailed and/or summary tables for each item (record) in the logical master file. (For example, a list of names, addresses and salaries for employees in the personnel file, with salary totals.)
2. Detailed and/or summary tables for a single or a group of related sets within the logical master file. (For example, a list of names, addresses, salaries and average age for employees by length of employment; with length of employment subset into one year, two to five years, over five years of service.)
3. Detailed and/or summary tables for two dimensional cross-tabulations of groups of related sets. (For example, number of employees, average age, average salary by length of employment (as above) versus professional level; with professional level subset into clerical, administrative, junior scientist, senior scientist, executive.)

The ANALYZER provides an **OUTPUT** command which allows the user to reorganize and consolidate the analysis steps performed on a file. **OUTPUT** can be used to build subsets of the existing files to make using the ANALYZER more efficient by having it operate only with the data upon which the user wishes to focus. **OUTPUT** allows the conversion of "virtual" sets into "actual" field values so that ANALYZER results can be used with other ADMINS commands, or handed off to statistical packages. (For example, code the sets for low, medium and high career mobility into an actual field. These sets could have resulted from an analysis of promotion and salary history.)

In summary, the ANALYZER maximizes the user's productivity in analyzing data without requiring a technical grasp of ADMINS or a professional assistant to direct and manage the computer processing. The ANALYZER is the user's "assistant". The ANALYZER focuses on providing a user oriented environment, including on-line syntax assistance, naming and management of derived fields and sets, automatic formatting of reports and graphs, and production of analytical results in further usable ADMINS data files and external file formats.

12.1.2 Conventions and Concepts

Before describing the syntax of the various ANALYZER commands, some conventions and concepts used in this manual are introduced.

1. A SET is an ordered list of item (record) numbers from the master file. (The numbers are the record positions of the records in the file.) A set has a short name and may have a descriptive label for use in reports. The set name must be one continuous string, i.e. a set name does not contain imbedded blanks, but may contain other punctuation characters such as "." or "-". Sets are specified in command syntax as "S:set-name" although the "S:" is only required where it is needed to resolve ambiguity in the command syntax.
2. The set "label" may contain blanks and may be up to sixty characters in length. A label can be designated on the same line as the set name or can be assigned to a set name subsequent to naming the set. Labels are used to describe columns or rows in reports. Set names and labels can be changed at any time, using the NAME command (Section 12.13.2 "Using the NAME Command With Sets").
3. In cases where it is necessary to distinguish a field name from a constant or label, an "F:" precedes the field name. The "F:" syntax is used in the SELECT and NAME commands (Section 12.4.1 "Single Set Syntax" and Section 12.13.1 "Using the NAME Command with Fields").
4. The notation "abc*" refers to all set names that start with characters abc. (abc can be any number of letters.) That is, the "*" is a "wild card" symbol. Wild cards are supported in the TABLE, GRAPH, and SHOW commands.

   Wild cards are a valuable tool for creating groups or classes of sets that are placed in reports or graphs as a unit. The wild card technique can be used to support simple hierarchical classifications of sets.
5. When the ANALYZER expects the user to type a command, it prompts with "AN>".
6. In the examples in this manual, the computer prompts appear in upper case, and the user's responses appear in lower case.
7. In the syntax descriptions used below square brackets (e.g. [ ]) around a syntax element are used to indicate that the particular syntax element is optional. For clarity the ANALYZER command names in this manual are fully spelled. However, the ANALYZER commands usually may be abbreviated to their initial 2 letters.
8. To specify any existing field name or set name it is sufficient to type only enough characters to unambiguously identify the field or set.
9. When using the ANALYZER, a list of all the set building and definitional commands are saved in a file named XXX.SAV. Thus, one can leave an ANALYZER session and later return to where the work was left. SAV files are described in Section 12.2.2 "The SAV File".
10. The ANALYZER provides syntax assistance by simply typing the command name and a carriage return. For the SELECT, TABLE, and GRAPH command, type the command name followed by two carriage returns to see the syntax assistance. The ANALYZER also has on-line HELP, which describes the purpose as well as the syntax for each command. The on-line ANALYZER HELP is describe in Section 12.15 "The HELP Command".

12.1.3 Command Summary

The following commands are supported in ANALYZER.

Definitional

FILE	Names the master file.
LINK	Asserts the link file, the keys to them, and the link fields
CREATE	Create new fields based on calculations or logic, using same syntax as ADMINS report.
NAME	Changes the name and/or assigns or changes the label for a field or set.
OPTION	Sets various printing and reporting options.

Set Operations

SELECT	Uses comparisons to build first level sets.
MARG	Automatically builds first levels sets.
INTERSECT	Logical “and” of two or more sets.
UNION	Logical “or” of two or more sets.
COMPLEMENT	Logical negation of a set.
EXCLUSIVE OR	Exclusive “or” of two sets.

Output

TABLE	Produces detail or summary reports.
GRAPH	Graphs values from a summary report.
OUTPUT	Outputs records to an ADMINS data file from actual fields, linked fields, created fields, and sets.
WRITE	Writes a command file.

Informational

SHOW	Displays field, set or file descriptions
EXAMINE	Shows a full step by step breakdown of a set.
HELP	Assists user.
STOP	Exits from the ANALYZER, saves latest changes.
QUIT	Exits from the ANALYZER, does not save latest changes.

Inter-Process

SPAWN	Create subprocess.

12.1.3.1 Recall & Line Editing in the ANALYZER Session

During ANALYZER interactive sessions there is often a need to display a previously entered command to fix a typo, make a change, or simply to re-use the command, instead of retyping the whole line over again. The ANALYZER saves and allows you to access up to 20 most recently entered commands.[2] Once you have recalled a command, you can use several editing keystrokes supported by the ANALYZER to facilitate changing the command.

Use the UP arrow key to recall previously entered commands, and use the DOWN arrow key to re-examine a command previously retrieved using UP arrow.

You may then edit the ANALYZER command line using the following keystrokes:

Edit Key	Function
CTRL/A or Insert-Here	Switches between overstrike and insert mode. The default mode is overstrike.
LEFT arrow or CTRL/D	Moves the cursor one character to the left.
RIGHT arrow or CTRL/F	Moves the cursor one character to the right.
CTRL/H or F12	Moves the cursor to the beginning of the line.
CTRL/E	Moves the cursor to the end of the line.
CTRL/U	Deletes characters from the beginning of the line to the cursor.

2. Recall and line editing are **not** supported in window mode.

12.1.4 The Preventive Maintenance Example

Throughout the rest of this document, we will refer to a preventive maintenance study of 40 vehicles which was conducted in 1981. The file to be analyzed contains the following fields.

```
************** Preventive Maintenance File **************
MAS 100
VEH#     X9999 KEY1   "vehicle identification number"
MODYR    I            "model year, last 2 digits of year"
TOTMILES D            "total miles"
MILES    D            "miles driven in 1981"
CLASS    I            "weight class, 1-5"
TOTMAINT D2           "total maintenance cost"
PARTS    D2           "cost for parts in 1981"
LABOR    D2           "cost for labor in 1981"
```

The data for this file is as follows. Whenever you read an example, you can refer to this data listing and see how the ANALYZER performed a particular operation.

VEH#	MODYR	TOTMILES	MILES	CLASS	TOTMAINT	PARTS	LABOR
0001	76	36,456	12,762	5	3,465.54	132.00	231.50
0002	71	87,434	9,834	3	5,384.53	543.00	654.00
0003	73	38,004	5,476	5	5,378.00	.00	546.32
0004	75	54,538	7,623	1	4,648.00	546.12	675.00
0005	77	34,879	12,453	1	4,125.00	544.00	179.00
0006	79	15,437	7,634	3	1,437.76	345.76	578.00
0007	80	12,098	5,643	3	899.00	357.00	368.00
0008	81	7,234	3,425	4	749.00	127.00	433.00
0009	82	3,265	3,265	2	200.00	125.00	75.00
0010	73	125,987	32,555	4	4,855.00	234.00	765.80
0011	74	98,555	12,876	1	3,986.00	433.00	578.00
0012	76	58,098	13,564	5	3,122.00	400.00	350.00
0013	78	37,324	14,324	1	1,943.00	123.78	657.00
0014	80	14,786	7,456	1	1,080.00	345.00	455.00
0015	81	9,655	9,655	3	875.00	490.00	385.00
0016	69	150,886	21,376	2	7,311.00	577.00	308.00
0017	72	76,945	19,745	3	5,070.00	694.00	374.00
0018	74	105,886	16,543	5	1,044.00	487.00	75.00
0019	74	54,654	14,354	2	3,887.00	345.00	945.00
0020	74	86,442	12,543	5	4,872.00	497.00	855.00
0021	77	67,444	9,834	4	2,509.00	254.00	199.00
0022	79	43,765	11,546	1	4,886.00	627.00	362.00
0023	81	7,544	7,544	2	1,055.00	675.00	480.00
0024	80	7,634	3,564	1	456.00	211.00	190.00
0025	73	87,654	14,543	2	2,077.00	546.00	981.00
0026	75	75,330	12,546	1	3,576.00	263.00	946.00
0027	76	65,435	15,436	2	4,099.00	625.00	414.00
0028	78	65,333	12,088	4	3,462.00	524.00	836.00
0029	79	31,234	8,976	3	2,763.00	526.00	684.00
0030	79	23,734	8,456	4	1,077.00	376.00	132.00
0031	77	63,000	8,000	3	6,565.00	543.00	1,777.00
0032	80	17,000	7,500	2	675.00	400.00	275.00
0033	81	10,000	10,000	2	200.00	80.00	120.00
0034	78	44,000	12,000	4	2,050.00	200.00	420.00
0035	75	70,000	15,300	5	5,365.00	364.00	955.00
0036	69	120,000	8,000	5	15,080.00	600.00	150.00
0037	80	35,000	17,000	4	2,500.00	250.00	400.00
0038	79	25,000	7,500	3	900.00	100.00	200.00
0039	76	60,000	15,000	4	2,000.00	400.00	600.00
0040	82	67,000	10,500	5	1,940.00	290.00	175.00

Although the ANALYZER is capable of performing complex analyses, we have tried to keep the examples simple, focusing on simple illustrations of the features, rather than "real life" analysis techniques.

12.1.5 Personnel File Example

Some of the examples used in this manual refer to a personnel file containing fringe benefit budgeting information. The file definition of this personnel file is included here for reference.

```
********* PERSONNEL FRINGE BENEFIT BUDGETING **********
MAS 500
EMPL#       X999999    KEY1
LNAM        A20                "Last name"
FNAM        A20                "First name"
ADDR        A30                "Address"
CITY        A20                "City"
STATE       A2                 "State"
DEPT        X9999              "Department number"
SLOT        X99999             "Position"
JCLASS      X9                 "Job Class"
HIREYR      I                  "Year Hired"
SEX         A2                 "Sex"
DOB         DA                 "Date of Birth"
MARIT       A1                 "Marital Status"
EDUCATION   A2                 "Education level"
SALARY      D2                 "Total Salary"
PENSION     D2                 "Actual Pension Amt"
ASS         D2                 "Actual Social Security Amt"
AHOSP       D2                 "Actual Hospitalization Amt"
UNION       A5                 "Union # or exempt"
```

12.2 The FILE Command and SAV Files

This section discusses the FILE command and SAV files. The FILE command is used to name the ADMINS data file to be analyzed. SAV files are automatically maintained by the ANALYZER for the purpose of keeping track of the analysis steps. Because the FILE command and SAV files are interdependent, these two concepts are discussed together in this section.

12.2.1 The FILE Command

The analysis session begins by typing "AN" to the operating system prompt. The FILE command is used to identify the ADMINS data file to be used as the master file in the analysis. As one would expect, this is typically the first instruction the ANALYZER receives.

```
AN> file file-name [/RANGE]
```

In the following example, the file PREV.MAS, which contains the vehicle maintenance data, is named as the master data file, right after the user enters the ANALYZER.

```
$ an
AN> file prev.mas
```

Alternatively, the master file can be included on the AN command line, so that the FILE command is not necessary. For example:

```
$ an prev.mas
```

It should be clear that without a master file on which to work, there is little that the ANALYZER can be told to do. Hence, identifying the master file is almost always the first step in using the ANALYZER.[3]

The RANGE qualifier allows you to define a virtual file limited to a key range (the file must be in sort order). RANGE prompts the user for the beginning and the end of the key range. If the main file has multiple keys, enter some or all of the key values, separated by blanks. Null values are used for any minor keys not entered. Range is useful when operating on a large data file when only part of it needs to be analyzed. This speeds up performance significantly.

The RANGE qualifier can only be used with a file name, either in the FILE command or on the AN command line:

```
$an
ANALYZA1.SAV CREATED
AN>file prev.mas/RANGE
Start of key range: KEY_VALUE(s) or ?
End of key range..: KEY_VALUE(s) or ?
```

or alternatively,

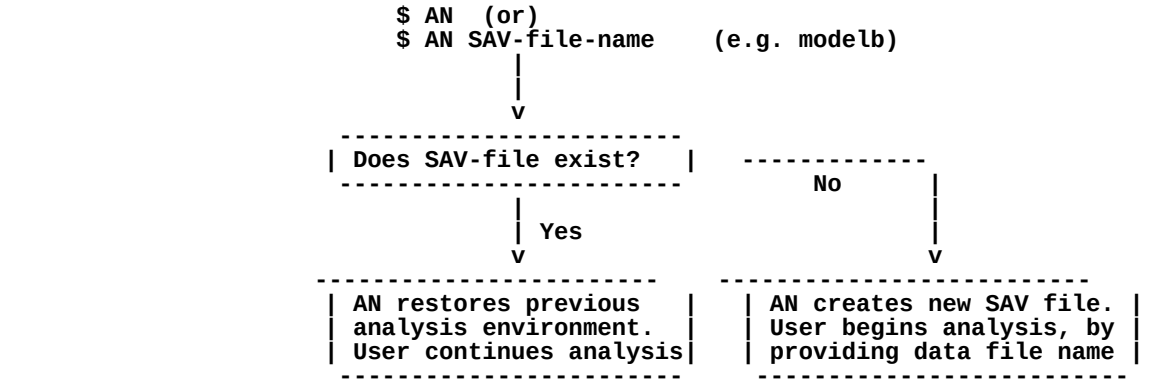
```
$an prev.mas/RANGE
ANALYZA1.SAV CREATED
Start of key range: KEY_VALUE(s) or ?
End of key range..: KEY_VALUE(s) or ?
```

Replying to the range prompts with a question mark (?) displays the file's key fields and their field types.

12.2.2 The SAV File

The ANALYZER automatically maintains a log of all the definitional and set building commands used in the analysis session in a SAV file (e.g. ANALYZB3.SAV). That is, the ANALYZER always maintains the ability to re-create all of the instruction steps that brought the analysis to its current point. One can return to an analysis started previously, simply by typing the AN command. The ANALYZER then restores all of the steps which were used and saved in the default SAV file, and then allows the user to continue the analysis.

The following diagram illustrates what happens when an analysis session begins.



3. The ANALYZER can be asked to show the field names in an ADMINS data file, before a master file is selected as the main file for the analysis. This use of the SHOW command is described in Section 12.12.3 “Show File-Name”.

If the user simply types "AN" to the operating system prompt, the default SAV file (e.g. ANALYZB3.SAV) is automatically used.

New SAV files have either a default name or a user specified name. To create or use SAV file, with a name other than the default, include the name of the SAV file on the AN command line. The file type (.SAV) is not included on the command line. For example, to begin an analysis using a SAV file named MODELB.SAV type:

```
$ an modelb
```

To resume working on an existing **named** SAV file, the name of the SAV file is once again included on the AN command line as in the example above.

Regardless of the SAV file name, the first time a non-existent SAV file is needed by the ANALYZER, it is created. Subsequent use of that file will recall all of the commands which had been added to the SAV file during its previous usages.

The FILE command is used with a **new** SAV file, either a named SAV file or a new default SAV file. There can be only **one** data file named with the FILE command within an analysis session. (Additional data files can be linked as described in Section 12.3 "The LINK Command")

The contents of the SAV file can be written into a text editable format for future use, by using the WRITE command (see Section 12.18 "The WRITE Command").

If an ANALYZER SAV file is being reused and the ANALYZER detects that the records have been removed or added to the master file (i.e. the number of records or the last record position has changed) since its previous use, the ANALYZER will refuse to use the SAV file. But, it will automatically write a command file called SAVESAV.COM before it exits. By running the ANALYZER with this command file, the user is able to reconstruct the analysis steps that were in the SAV file on the updated data file.

A SAV file built before records have been removed from or added to the master file, is no longer usable. This is because the SAV file stores sets as a collection of pointers to the records in the master file. When a SAV file is reused, the sets are not rebuilt. Rather the ANALYZER uses the pointers to the records in a set as stored in the SAV file. Once records are added to or removed from the master file, the record pointers in the SAV file are no longer considered to be accurate. Therefore, the user can rebuild the sets for the altered data file, by running the command file SAVESAV.COM.

To reconstruct the SAV file using SAVESAV.COM, enter the following:

```
$ an @savesav
```

Note that the ANALYZER is able to reuse a SAV file after the data in an **existing** record in the master file has been changed. However a change to an existing record may indeed cause changes to set membership criteria of which the ANALYZER is unaware. In general, a SAV file should not be used after **any** change has been made to the master file.

Also the ANALYZER is able to reuse a SAV file after records have been added or removed from a **link file.** It is the user's responsibility to reconstruct the SAV file if either of these conditions cause the analysis to be out of step with the actual data.

12.2.2.1 Initializing the SAV File

There are three ways to initialize a SAV file. Once a SAV file has been initialized, all of its previous contents are lost. Therefore, one must be sure not to inadvertently initialize the SAV file.

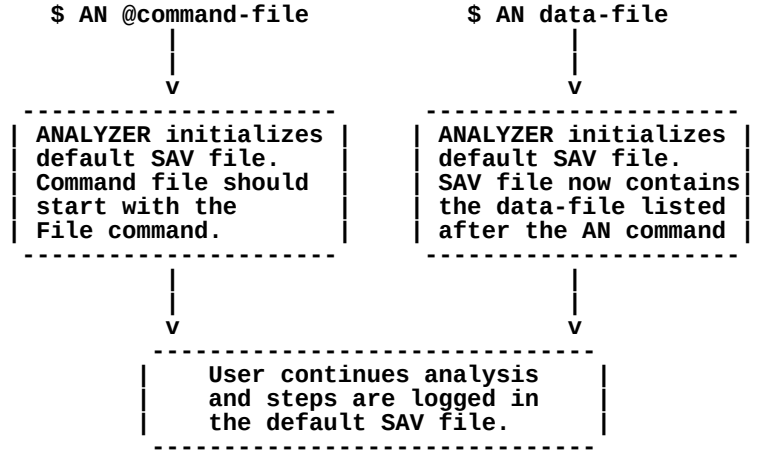
1. If the data file is included on the AN command line, then an existing default SAV file (e.g. ANALYZA7.SAV) is initialized. For example, by naming the data file, PREV.MAS on the AN command line, the default SAV file is initialized, and contains only the file name included on the command line.

```
$ an prev.mas
```

2. If an ANALYZER command file (see Section 12.21 “Command Files”) is included on the AN command line, then the default SAV file is initialized. Note, the command file name is preceded with the @ symbol. If the file type is not included, the ANALYZER will look for a file type of ".COM". For example, if ANALYSIS.COM and PREPARE.PRP are ANALYZER command files:

```
$ an @analysis

$ an @analysis.com

$ an @prepare.prp
```

The following is a schematic diagram which illustrates the first two ways to initialize the default SAV file.



3. The ANALYZER does not automatically initialize a "named" SAV file. Deleting the "named" SAV file has the same effect, i.e. the ANALYZER session starts "initialized".

```
$ delete modelb.sav;1
```

A default SAV file can also be deleted to start the ANALYZER session from the beginning.

12.2.3 Accessing Data Views

To access a Data View defined specified using the ADMINS Data Dictionary (see Appendix I: "ADD: The ADMINS Data Dictionary") append the qualifier **"VIEW"** after the name of the data view,[4] either on the command line, or with the FILE command:

```
$ an invoices/view

 or

$ an invoices -view


$ an
AN> invoices/view
```

Data Views are accessed in ANALYZER LINK commands (see Section 12.3 "The LINK Command") by simply providing a Data View name in place of a file name.

12.3 The LINK Command

The LINK command is used to link to records in files other than the file named with the FILE command to include fields from these link files in the logical master file. The LINK command names a link file and the fields from the logical master file that should be used to form the key into the linked file. After the link is defined the fields from the link file are treated as part of the logical master file.

Links can have several purposes in the ANALYZER.

1. Links may be used to relate records in two files which are really about the same item. (E.g. last year's budget by line item versus this year's.)
2. Links can relate many records to one record in another file, as in the case of linking from codes in one data file to their descriptions in a code data file. (E.g. linking department codes to their descriptions.) Also links can be made to data files that contain attributes for the codes that will be used to select particular items. (E.g. linking employee records to the department record in which they work to obtain departmental information about the employee.)
3. Links can also be for the purpose of cross referencing, i.e. to relate records from two files via a cross reference file. (E.g. linking records in a file about vehicle owners to records in a file about their vehicles.)

Values brought in via one link can be used to form another link.

4. The /VIEW qualifier is needed to avoid ambiguity. If the character string provided on the command line or in the FILE command does not contain a period (.) the ANALYZER treats it as the user-specified name for a save file to be created or re-opened.

12.3.1 LINK Syntax

The syntax for the LINK command follows.

```
AN> link [nomem/mem] fields [or all] from file-name key is field(s)
```

If there is a naming match between fields already in the logical master file, and the new fields being added to the logical master file via the LINK command, then the ANALYZER will prompt to rename the matched field. If the new field is not renamed, i.e. the user presses ENTER to the "RENAME or SKIP:" prompt, the new field will not be added to the logical master file.

When entering the link field names, individual fields in the link file can be listed, separated by a blank. Alternatively, ALL can be used to link in "all" fields from the link file.

Note that links are performed **before** created fields are calculated. Therefore, created fields cannot be used to create a key for a link field. See Section 12.16 "The CREATE Command" for a discussion of created fields.

Examples of the link syntax follow:

```
AN> link yracq from register.mas key is veh#

AN> link all from violation.mas key is license

AN> link nomem all from model.tab keys are make mod#
```

12.3.2 Efficiency Considerations

By default, the LINK command tries to read the entire link file into memory, to make the link processing more efficient.[5] If there is insufficient space in memory for the link file, then links will be performed to disk. The user can instruct the ANALYZER not to try to load the link file into memory using the NOMEM option.

The ANALYZER only performs the link when another command (e.g. TABLE, INTERSECT, or OUTPUT) really requires the use of that field.

To summarize, there are two ways links can be processed.

1. Directly in memory. This is typically used for **small** link files and is very efficient. The ANALYZER attempts to load link files into memory by default.
2. The more general case, when the ADMINS data file key search techniques are used to fetch the link records. An alternative to this approach is to use the OUTPUT command (see Section 12.17 "Output Files") to derive a file placing all the fields of interest into one physical file.

5. By default the maximum number of records that the ANALYZER will read into memory is 2000. The system manager can vary this value by assigning a numeric value to the SYSTEM logical name ADM$LNKMEM, i.e. "ASSIGN/SYSTEM 5000 ADM$LNKMEM".

12.4 SELECT

The SELECT command is used to build sets by reading through the logical master file (i.e. the universe of items), and applying the criteria supplied by the user to build sets. SELECT can be used to give the ANALYZER instructions for building multiple sets at one time. This is more efficient than building one set at a time because multiple sets can be built with only one pass through the logical master file. The SELECT command can also be used to build sets with respect to the items in an existing set, rather than the whole logical master file.

12.4.1 Single Set Syntax

The syntax for building one set at a time follows.

```
AN> select [s:set-name] criteria [s:set-name [label]]
```

The "criteria" consist of a syntax of form:

```
field operator constant/f:field
```

The comparison operators used by the ANALYZER are:

Operator	Function
EQ	equal to
NE	not equal to
LE	less than or equal to
GE	greater than or equal to
LT	less than
GT	greater than
BET	between
IN	includes (alphanumeric and text fields only)

EQ and NE can be compared to multiple values. In this case the implied syntax is "EQ A or B ..." and "NE A and B ...". For example, "SELECT CLASS EQ 1 2" means "select class is equal to 1 or 2", or "SELECT MODYR NE 79 80 81" means "select model year not equal to 79 and 80 and 81".

If a field is compared to another field, (e.g. "SELECT PARTS GT F:LABOR") the field to the right of the operator is preceded by "F:" to indicate that LABOR is a field name rather than a constant.

The "S:SET-NAME" (at the end of the line) is optional in this syntax to allow the user to name and save the set, or to discard the set **after** seeing how many items are in the set (the "item count"). Hence if no set name is given, SELECT prompts:

```
n items, S:
```

after building the set. The user can type a name (and optionally a label) or simply press ENTER to tell SELECT **not** to keep the set. Whether or not a set name is supplied on the SELECT line, SELECT shows the item count after the set is built.

If the set is named (and therefore saved), the SELECT displays the sequential set number, the set name, the item count and the percentage of the master file which this set represents.

The following examples illustrate the single set syntax for the SELECT command. The fields PARTS and LABOR contain the amounts (in dollars and cents) spent on repairing and maintaining the vehicles in the file for one year. Below, we build a set of vehicles with the cost of parts greater than $300.

```
AN> select parts gt 300
26 ITEMS, S:hiparts
1  S:HIPARTS  26   65%
```

The set HIPARTS is built which contains 26 records, representing 65% of the total file (40 records). Next we build a set a vehicles in which the labor cost were greater than or equal to $300.

```
AN> select labor ge 300 s:hilab
2  S:HILAB    27   67%
```

The set HILAB is built and named before the item count is displayed.

The IN (includes) operator is used to search for character strings in alphanumeric fields. Both of the following build a set of employees with a first name (FNAM) that includes the string "Barbara".

```
AN> select fnam in 'Barbara'

        or

AN> create checknm/a7 'Barbara'
AN> select fnam in f:checknm
```

Note that the IN (includes) operator is **case sensitive when used with alpha fields,** but **case insensitive when used with text (TXnn or TInn) fields.**

12.4.2 Multiple Set Syntax

In the multiple set or "batch" SELECT syntax, the set name is usually supplied on the same line with the criteria for that set. If the set name is omitted, SELECT prompts for a set name, e.g. "S:". Set labels are always optional, and can be added later to the set name.

```
AN> select [s:set-name]
SELECT> criteria s:set-name [label]
SELECT> criteria s:set-name [label]
...
SELECT> cr
```

SELECT keeps reading for criteria until the user presses ENTER to the SELECT> prompt. Then all the sets are built in one pass through the file. SELECT prints a sequential number for each set name with the number of items in the set, and the percentage of the master file.

The following example illustrates the multiple set syntax for the SELECT command. The field MODYR contains a 2 digit model year for the vehicles. The field TOTMAINT contains the total maintenance costs, and the field TOTMILES contains the total miles for the vehicles.

```
AN> select
SELECT> modyr eq 81 s:y.81
SELECT> modyr eq 80 s:y.80
SELECT> modyr eq 79 s:y.79
SELECT> totmaint lt 1000 s:lomaint
SELECT> totmiles gt 50000 s:himile
SELECT> cr
```

12.4.3 SELECT Using An Existing Set

The SELECT command can be used to select items by examining the items contained in an existing set, rather than examining the entire logical master file. Instead of reading through the entire logical master file to build the sets, SELECT only reads those items of the logical master file that are present in the specified set. This technique is a more efficient way to analyze a subset of a large master file.

Using SELECT with respect to an existing set can be done with either single or multiple set syntax. In either case the set name to be used in the sub-setting directly follows the SELECT command. If single set syntax is used, the SELECT criteria follow the set-name on the same line. For example:

```
 AN>select s:setname field operator constant/f:field
[s:setname[label]]
```

If the multiple set syntax is used, enter the SELECT command and set name followed by pressing ENTER. At the SELECT> prompt, the rest of the syntax is the same as when no set-name is used on the command line. For example:

```
AN> select s:set-name
SELECT> criteria s:set-name [label]
SELECT> criteria s:set-name [label]
SELECT> ...
SELECT> cr
```

In the following example, a set is built from the set HIPARTS (PARTS GT 300) in which the vehicles are from '76 or older.

```
an> select s:hiparts modyr le 76
5 ITEMS, S: oldhipart
```

In the following example, the items in the set HILAB (LABOR GE 300) are used to build sets based on the weight class (CLASS) of the vehicles.

```
AN> select s:hilab
SELECT> class eq 1 s:light
SELECT> class eq 2 3 s:med
SELECT> class eq 4 5 s:heavy
SELECT> cr
```

12.5 The MARGINAL Command

The MARGINAL command (MARG) is used to have the ANALYZER automatically build first level sets either for coded data or numerical values. MARG generates the first level sets for one or several fields automatically, by reading through the logical master file and making a list of unique values for each field requested. Then MARG creates SELECT instructions, which the ANALYZER executes, to build sets to each of these values. After the MARG command is given, the process proceeds automatically without user intervention.

12.5.1 MARGINAL Syntax

The syntax for the MARG command is:

```
AN> marg [s:set-name] field1[/n] [field2[/in]] ...
```

If a field contains coded data (e.g. position level or survey response) the "/n" notation is used to build sets for the "n" most frequently occurring values. One additional set is built containing all the other values. "N" can be a value between 2 and 249. If "/n" is omitted it is assumed to be "/10". That is, sets are built to the "n" most frequent values, and an "n+1-th" set is built to all the other values.

If a field contains numeric data (e.g. budgets or salaries), MARG builds sets for "n" intervals of values using the "/in" syntax as described in Section 12.5.2 "Using MARG to Build Interval Sets" below.

In the example which follows, the MARG command is instructed to build sets for the codes in the field SEX, as well as for the four most frequent codes for the field EDUCATION. MARG automatically builds a set for the codes in of the field EDUCATION not included in the first four EDUCATION sets. (In this example SEX only contains F or M.)

```
AN> marg sex education/4
400 RECORDS READ
    1    S:SEX.1           195    54%     SEX:F
    2    S:SEX.2           163    45%     SEX:M
    3    S:EDUCATION.1       8     2%     EDUCATION:PHD
    4    S:EDUCATION.2      80    20%     EDUCATION:MA
    5    S:EDUCATION.3     120    30%     EDUCATION:HS
    6    S:EDUCATION.4     160    40%     EDUCATION:BA
     7    S:EDUCATION.5      32     8%     EDUCATION:_OTHER
```

MARG names the sets by appending a sequential number to the field name (or abbreviated field name) supplied on the MARG command line. Each set is automatically given a label describing the code used to create that set. This set naming convention provides easy use of wild card notation when specifying set names for TABLES.

12.5.2 Using MARG to Build Interval Sets

If the field is a date or numeric type (integer, decimal or four word decimal), then MARG may be instructed to build "interval" sets, using the "/in" notation. Interval sets divide the items into halves, quarters, tenths, etc. (i.e. "n" parts) based on the numerical values in the requested fields. MARG is instructed to build interval sets by appending "/i" to the field name. The number of interval sets to be built may (optionally) be specified by the number (n) which the user types following the "/i". The value of "n" may be between 2 and 249. If "n" is omitted, MARG attempts to build 10 interval sets.

In the following example, the MARG command is used to build sets for four salary range intervals. Then we compare average salaries for men and women and display the results in a table. (The TABLE command is described in Section 12.10 "The TABLE Command".) We use the wild card notation to request all of the SALARY sets. Note, that in this example, the SALARY intervals sets are not equal in size, that is the population could not be divided into exact quarters. This is because MARG encountered multiple records with the same values. In this case MARG builds interval sets as equally sized as possible.

```
AN> marg salary/i4
358 RECORDS READ
    8    S:SALARY.1/4      92     25%    SALARY:5,020-11,640
    9    S:SALARY.2/4      95     26%    SALARY:11,641-13,457
   10    S:SALARY.3/4      89     24%    SALARY:13,458-18,439
   11    S:SALARY.4/4      82     22%    SALARY:18,440-


AN> table
ACROSS THE TOP: sex.1 sex.2
DOWN THE SIDE: salary.*
VALUES: salary/av #int %tot

                          SEX.1 (195)                SEX.2 (163)
                             SEX:F                      SEX:M
                  SALARY/AV  #INT %TOT    SALARY/AV  #INT %TOT
SALARY.1/4 (92)
SALARY:5,020-11,640        9,856    56  15%       9,789    36  10%
SALARY.2/4 (95)
SALARY:11,641-13,457      12,671    59  16%      12,700    36  10%
SALARY.3/4 (89)
SALARY:13,458-18,438      15,613    46  12%      16,133    43  12%
SALARY.4/4 (82)
SALARY:18,439-43,900      23,195    34   9%      24,231    48  13%
```

We notice that there are 48 males (or 13% of the employees) as compared to 34 females (or 9% of the employees) in the highest salary quartile. Also, the average salary for men in the highest salary range is over $1,000 higher than the average salary for women in this group.

12.5.3 MARGINAL Using An Existing Set

Like the SELECT command, the MARG command can be instructed to build sets based on a subset of the master file. In this case MARG only operates on the items in that set, rather than examining the entire logical master file. For example:

```
AN> marg s:sex.1 maritstat
```

This example tells MARG that within the set named SEX.1, (which is equivalent to "SEX EQ F"), build sets to the different marital statuses.

Logically, building a set on a subset of the file (instead of the entire logical master file) is analogous to using the INTERSECT command (see Section 12.6.1 "INTERSECT Syntax").

12.6 The INTERSECT Command

The INTERSECT command "intersects" two or more sets and places the resulting items in another set. Intersection consists of placing in the output set any item that was present in **all** of the two or more input sets.

12.6.1 INTERSECT Syntax

The INTERSECT command accepts two or more set names for input sets. The output set name is optional because the user may wish to decide whether or not to keep the output set after the user sees how many items result in the set. If the output set is named on the INTERSECT command line, the equal sign (=) is required.

```
AN> intersect set-name1 set-name2 ... [= new-set-name [label]]
```

Unlike the SELECT or MARG commands which must first read through the data before creating the set, the size of the new set created by the INTERSECT command is viewed immediately. This is because unlike SELECT or MARG, INTERSECT (and UNION, COMPLEMENT, and EXCLUSIVE OR) are performed very rapidly because the ANALYZER **does not read** through the logical master file to perform set operations. The ANALYZER works directly with the sets in question.

For example the set HICOST is built with all of the items (vehicles) that are in **both** the sets HIPARTS (26 vehicles with high cost for parts) and HILAB (27 vehicles with high cost for labor).

```
AN> intersect hiparts hilab = hicost
   13      S:HICOST        18     45%
```

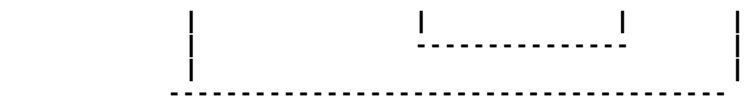
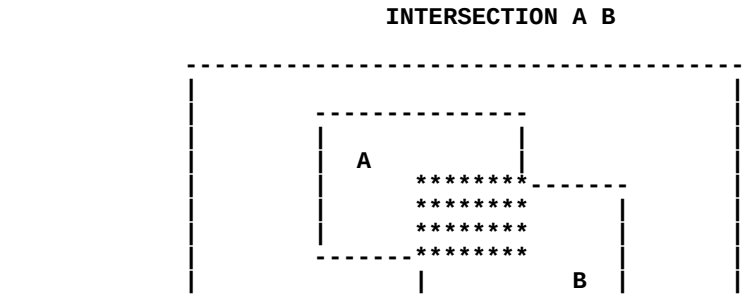


Figure 1: Intersection Diagram



The result of an INTERSECTION is the "shaded area", that is the items present in Set A **and** Set B.

12.7 The UNION Command

The UNION command is used to create a set which contains **all** of the items in any of its two or more input sets.

12.7.1 UNION Syntax

UNION uses the same syntax and usage rules as INTERSECT.

```
AN> union set-name1 set-name2 ... [= new-set-name [label]]
```

The difference between INTERSECT and UNION is in the logic applied by UNION in selecting items to be placed in the output set. Whereas INTERSECT selects only those items present in **all** of the input sets, UNION selects those items present in **any** of the input sets. As one would expect, INTERSECT produces sets smaller than or equal to its smallest input set, whereas UNION produces sets larger than or equal to its largest input set. In either case an item is represented in the output set only **once.**

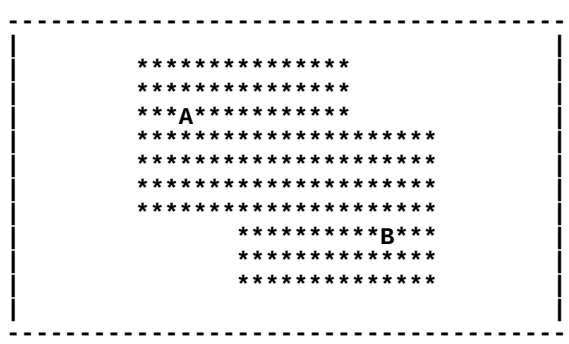
In the following example, the set 79HILAB is built with **all** of the items (vehicles) in HILAB (high labor costs) or in Y.79 (model year 1979) or in both of these input sets. The new set, 79HILAB, is named on the command line.

```
AN> union hilab y.79 = 79hilab
```

In the next example, the set NEW is built with all of the items in the sets Y.81 (model year 1981), Y.80 (model year 1980) or Y.79 (model year 1979). Note that the set NEW is named after the item count was displayed.

```
AN> union y.81 y.80 y.79
 10 ITEMS, S: new
```

Figure 2: Union Diagram



The result of the UNION is the "shaded area", that is the items present in either Set A **or** in Set B **or** in both.

12.8 COMPLEMENT

The COMPLEMENT command creates a set which is the negation of another set, i.e. COMPLEMENT creates a set which contains items **not contained in** another set. The COMPLEMENT of a set can be with respect to the entire logical master file or with respect to another set.

12.8.1 COMPLEMENT Syntax

COMPLEMENT uses the same syntax structure as INTERSECT, except COMPLEMENT can only operate on **one** or **two** input sets.

```
AN> complement set-name1 [set-name2] [= new-set-name [label]]
```

12.8.1.1 One Set Syntax

In the one input set syntax COMPLEMENT builds an output set that contains all items from the logical master file that were **not** present in the input set. The equal sign is required if the new set name is included on the COMPLEMENT instruction line. For example the complement of the set HIMILE (high mileage vehicles) is the set LOMILE (low mileage vehicles).

```
AN> complement himile = lomile
```

The new set can be named after the item count displays. The complement of the set of low maintenance vehicles (LOMAINT) is the set of high maintenance vehicles (HIMAINT).

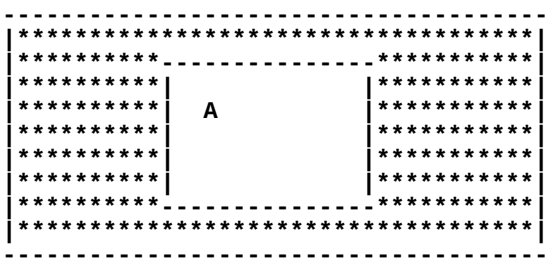
```
AN> complement lomaint
 12 ITEMS, S: himaint
```

12.8.2 Two Set Syntax

In its two input set syntax COMPLEMENT produces all items that are present in its second input set but are **not** present in its first input set. For example the set LOMILE.LIGHT (light weight low mileage vehicles) is built to contain all the items (vehicles) in the set LIGHT (light weight vehicles) that are **not** in the set HIMILE (high mileage vehicles).

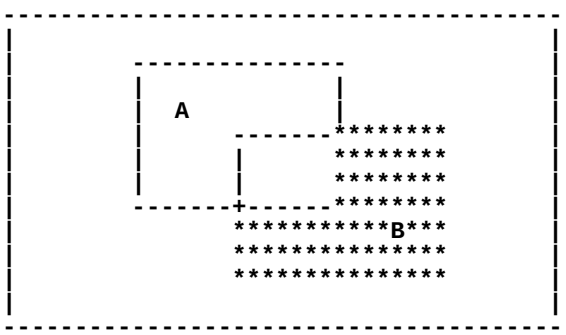
```
AN> complement himile light = lomile.light Low mileage light
vehicles
```

Figure 3a: Complement Diagram - One Set



The COMPLEMENT of Set A is the "shaded area", that is the items present in the logical master file but **not** present in Set A.

Figure 3b: Complement Diagram - Two Sets



The result of the COMPLEMENT of two sets. The "shaded area" represents the items in Set B (the second set) which are **not** present in Set A (the first set).

12.9 The XOR (Exclusive Or) Command

The Exclusive Or Command, XOR uses a two input set syntax and produces all items present in **only one** of its two input sets. XOR is the **"exclusive or"** operation in formal logic.

12.9.1 XOR Syntax

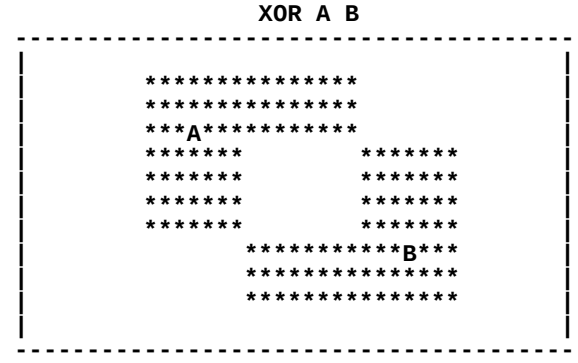
The Exclusive Or command is indicated by the letters XOR. The two input set names are required. The output set name is required only when saving the set.

```
AN> xor set-name1 set-name2 [= new-set-name [label]]
```

In the following example, we build a set of vehicles that are **either** in the HIPARTS set (26 vehicles) **or** in the HIMILE set (20 vehicles). That is, the output set does **not** contain vehicles that are both "hiparts" (the set of vehicles with parts costing over $300) and "himile" (the set of vehicles with over 50,000 miles).

```
AN> xor hiparts himile
 14 ITEMS, S: hiparts.or.himile
```

Figure 4: Exclusive Or Diagram



The result of the EXCLUSIVE OR of two sets is the "shaded area", that is the items in **only one** of the two input sets.

12.10 The TABLE Command

The TABLE command is the reporting tool in the ANALYZER. The form of a report is to place set names "across the top" and "down the side" of the page, and then place values and/or summaries of the items from the logical master file into the "cells" of the table formed by the "intersection" of the sets in the rows and columns. By default, TABLE displays its output on the user's terminal, to re-direct that output use the "OPTION LP" command (see Section 12.19.7 “Line Printer”).

12.10.1 TABLE Syntax

There are three variations of the TABLE syntax. Below are the syntax variations followed by descriptions and examples of each.

12.10.1.1 TABLE on the Whole Logical File

If the TABLE command is followed directly by field names, then ANALYZER prints these field names using the "whole file" as the set, i.e. displaying detail and/or total values for these fields in the logical master file.

```
AN> table field1[/op] [field2/op ...] [same]
```

The fields may include any of the fields in the original master file as well as created fields (see Section 12.16 “The CREATE Command”) and linked fields (see Section 12.3 “The LINK Command”). Field names may be abbreviated as long as the fields can be unambiguously identified. If the TABLE command is followed by the word SAME, the previously entered values for the TABLE command are used. Field operations (/op) are described in Section 12.10.2 “Operations On Values” below.

The following example displays the detailed values for the fields VEH#, PARTS, LABOR, TOTMAINT and CLASS for all the items in the file. The fields PARTS, LABOR and TOTMAINT, which are decimal field types, are automatically totaled at the end of the table (see Section 12.13.1 “Using the NAME Command with Fields”).

```
AN> table veh# parts lab totmaint class
VEH#        PARTS        LABOR     TOTMAINT  CLASS

0001       132.00       231.50     3,465.54      5
0002       543.00       654.00     5,384.53      3
0003          .00       546.32     5,378.00      2
 ...          ...          ...          ...
0040       290.00       175.00     1,940.00      5
     ------------ ------------ ------------
       15,199.66    19,783.62   127,566.83
```

12.10.1.2 TABLE on a List of Sets, One Dimensional

If set names[6] are included on the TABLE command line, then TABLE assumes that these sets will be **either** "across the top" **or** "down the side". For example,

```
AN> table set-name1 [set-name2 ...] [same]
VALUES: field1[/op] [field2/op %tot] [same]
```

If the TABLE command line includes the set names, TABLE assumes that this will be a one dimensional report on the sets named on the TABLE command line. TABLE does not prompt for "ACROSS THE TOP:" or "DOWN THE SIDE:" set names. Instead TABLE immediately prompts for "VALUES:" The user enters the field names to be displayed in the table. The fields may include created fields, linked fields as well as any of the fields in the original master file. %TOT may also be entered to the "VALUES:" prompt. This function is described in Section 12.10.3 "Int ersection Functions".

If the word "SAME" is entered to any of the TABLE prompts, the previously entered response to that prompt is used.

When the TABLE command is followed by set names, TABLE selects an output format based on the following criteria:

1. If only one set is requested, then the set is placed "across the top" of the page and detail values (data from each record in the logical master file) for the requested fields are displayed by default. Overriding the default of detail values is described in Section 12.10.2 "Operations On Values".
2. If multiple sets are requested, then the sets are placed "down the side" of the page and the default display is of total (aggregated) values.

In the following one dimensional report, detail values for the fields PARTS, LABOR and TOTMILES are displayed for the set HIMILE (high mileage vehicles). The set name HIMILE displays across the top of the page, followed by the item count (number of records in the set) enclosed in parentheses. Notice that totals also display for the fields PARTS, LABOR and TOTMILES, since they are numeric (decimal) fields. By default, decimal field types (Dn) automatically total in tables. See Section 2.4.2 "Field Data Types" for a discussion of field data types.

```
AN> table himile
VALUES: par lab totmil

              HIMILE (20)
    PARTS         LABOR      TOTMILES

   543.00        654.00        87,434
   546.12        675.00        54,538
   234.00        765.80       125,987
      ...           ...           ...
---------     ---------     ---------
 9,165.12     12,612.80     1,640,621
```

6. If a set name is also a field name, then the "S:" must precede the set name.

In the next example, total values for the fields TOTMAINT and TOTMILES are displayed for all of the set names beginning with "Y." (i.e. the sets built to specific model years). Since several sets are requested, the sets names display "down the side" of the page. Each set name is followed by the item count in parentheses. Notice that the requested field names have been abbreviated in these examples.

```
AN> table y.*
VALUES: totmai totmi

                TOTMAINT     TOTMILES
Y.81 (4)        2,879.00       34,433
Y.80 (5)        5,610.00       86,518
Y.79 (5)       11,063.76      139,170
```

12.10.1.3 Two Dimensional Table

In a two dimensional table, (i.e. a cross-tabulation) sets are placed both across the top and down the side. The values displayed in the cells are for the items in the **intersection** of the row and column sets displayed. These values may be totals, counts, averages, percentages, etc., based upon the items in the intersection of the row and column sets.

To create a two dimensional report, the TABLE command is followed by ENTER. Then TABLE prompts with "ACROSS THE TOP:" and "DOWN THE SIDE:". In each case TABLE expects the user to enter one or more set names or "SAME". (If SAME is used as a response to any of these prompts, TABLE will use the last response to the corresponding prompt.)

TABLE then prompts for "VALUES:", and the user enters names of fields from the logical master file, including actual fields, created fields, and linked fields. TABLE displays values based on these fields for the items in the **intersection** of the row and column sets.

```
AN> table
ACROSS THE TOP: set-name1 [set-name2 ...] [total] [same]
DOWN THE SIDE: set-name1 [set-name2 ...] [total] [same]
VALUES: field1[/op] [field2/op ...] [int function] [same]

Operations are: /D /V /E /AV /MA /MI /Sn /Rn /ME /SP /SD
                /%C /%R /%Cn /%Rn /%n
Intersection functions are: #INT %TOT %ROW %COL SIG ALL
```

In the following table, the totals for fields PARTS, LABOR and TOTMILES are displayed for the intersections of the sets HIPARTS and HILAB (high parts cost and high labor cost vehicles) with the sets HIMILE and LOMILE (high and low mileage vehicles).

```
AN> table
ACROSS THE TOP: hiparts hilab
DOWN THE SIDE: himile lomile
VALUES: par lab totmile
                  HIPARTS (26)                    HILAB (28)
           PARTS     LABOR  TOTMILES    PARTS     LABOR  TOTMILES
 HIMILE(20) 8,124.12 10,527.00 1,304,860 7,534.12 12,013.80
1,280,291
 LOMILE(20) 4,685.76  3,898.00   210,132 4,066.54  5,768.32
296,081
```

12.10.2 Operations On Values

When the user types the names of fields in response to the "VALUES:" prompt (or directly after the TABLE command) various field operations can be requested by appending a slash (/) and the operation to the field name. Multiple operations can be requested for a field. Each time a field is used successively for another operation, a ditto (") may be used instead of re-entering the field name. For example, PARTS/MI "/MA "/AV means display the minimum, maximum and average value for the field PARTS. The ditto means repeat the field to the immediate left.

The following list of simple aggregation and sorting operations can be appended to fields in any of the table formats except where noted.

Operation	Meaning
/D	Display the value for this field for each item in the set or in the whole logical master file. This is the default operation if the field is requested on the TABLE command line or if only one set is requested. /D may not be used when there are multiple sets across the top.
/V	Display total values for this numeric field. This is the default operation if multiple sets are requested or in a two dimensional table.
/E	Count the existences of non-zero (or non-blank) values for this field.
/AV	Display the average for this numeric field.
/MA	Display the maximum value for this field.
/MI	Display the minimum value for this field.
/Sn	Sort the table in ascending order on this field. This field is the nth sort criteria. /Sn displays detail values for each item.
/Rn	Sort the table in reverse (descending) order on this field. This field is the nth sort criteria. /Rn displays detail values for each item.
/ME	Median Value: the middle value (numeric only)
/SP	Standard Deviation of the population (n)
/SD	Standard Deviation of a sample (n-1)

In the following example, the minimum, maximum and average values for the field TOTMAINT are displayed for the set names beginning with "Y." When set names are requested using the wildcard (*), the sets display in the order in which they were built.

```
AN> table y.*
VALUES: totmaint/mi "/ma "/av

              TOTMAINT/MI  TOTMAINT/MA  TOTMAINT/AV
Y.81 (4)           200.00     1,055.00       719.75
Y.80 (5)           456.00     2,500.00     1,122.00
Y.79 (5)           900.00     4,886.00     2,212.75
```

A field name without any operation code defaults to "/D" if the field is included on the TABLE command line, or if there is only one set displayed in the table. If there are multiple sets across the top and/or down the side, then a field name without any operation code defaults to "/V", totals.

In the next example, multiple sets are displayed down the side. The cells represent the totals for the fields PARTS, LABOR, TOTMAINT and TOTMILES for each of the set names beginning with "Y.", i.e. the sets to three model years.

```
AN> table y.*
VALUES: par lab totma totmi

                PARTS     LABOR     TOTMAINT   TOTMILES
Y.81 (4)     1,372.00  1,418.00     2,879.00     34,433
Y.80 (5)     1,563.00  1,688.00     5,610.00     86,518
Y.79 (5)     1,974.76  1,956.00    11,063.76    139,170
```

In the next example, a two dimensional report displays the TOTMILES and the average TOTMILES for the intersection of the set HIPART with sets HIMILE and LOMILE. The ditto (") means repeat the field to the immediate left.

```
AN> table
ACROSS THE TOP: hipart
DOWN THE SIDE: himile lomile
VALUES: totmiles "/a

               HIPARTS (26)
              TOTMILES TOTMILE/AV
HIMILE (20)  1,304,860     81,553
LOMILE (20)    210,132     21,013
```

12.10.2.1 Percentages of Values

The operations which follow are used to display the field subtotals as a percentage of the column or row totals for that field, or as a percentage of the same field in another row or column, or as a percentage of another field in the same cell. These percentages are rounded to the nearest whole number.

Operation	Meaning
/%C	Displays the value as a percentage of the column total for that field. Automatically adds the “pseudo-set” TOTAL to the “down the side” list of sets.
/%R	Displays the value as a percentage of the row total for that field. Automatically adds the “pseudo-set” TOTAL to the “across the top” list of sets.
/%Cn	Displays the value as a percentage of that same field in the nth column set of the table.
/%Rn	Displays the value as a percentage of that same field in the nth row set of the table.
/%n	Displays that value as a percentage of the nth field in the same cell of the table. A cell includes all the field names requested at the “VALUES:” prompt.

In the two dimensional table below, the operations /%R and /%C are used to display subtotals for the field TOTMAINT as a percentage of the row set and the column set totals. Note, the "pseudo-set" TOTAL is not explicitly included in the set name listing. However TABLE automatically includes the "pseudo-set" TOTAL when the operations /%R and /%C are requested.

```
AN> table
ACROSS THE TOP: lomaint himaint
DOWN THE SIDE: y.*
VALUES: totmaint "/%r "/%c

          LOMAINT (8)              HIMAINT(32)                  TOTAL
    TOTMAINTTOT/%RTOT/%C  TOTMAINTTOT/%RTOT/%C  TOTMAINTTOT/
%RTOT/%C
 Y.81 1,824.00   63%   38%  1,055.00   36%    7%  2,879.00  100%   14%
 Y.80 2,030.00   36%   42%  3,580.00   63%   24%  5,610.00  100%   28%
 Y.79   900.00    8%   18% 10,163.76   91%   68% 11,063.76  100%   56%
 TOTAL4,754.00   24%  100% 14,798.76   75%  100% 19,552.76  100%
100%
```

As we see in the example above, the total maintenance (TOTMAINT) for the cell formed by the intersection of sets LOMAINT and Y.81 is 1,824. This is 63% of the row total (2,879) and 38% of the column total (4,754).

In the next example, the total value for PARTS maintenance is compared to TOTMAINT (total maintenance) for the sets built for 3 model years (1979-1981). Since the field TOTMAINT is the second field listed, PARTS/%2 displays the percentage of the field PARTS with respect to the total for the second field (TOTMAINT).

```
AN> table y.*
VALUES: parts totmaint parts/%2

                  PARTS     TOTMAINT PAR/%2
Y.81 (4)      1,372.00     2,879.00    48%
Y.80 (5)      1,563.00     5,610.00    28%
Y.79 (5)      1,974.76    11,063.76    18%
```

In the table above, we see that PARTS comprises 48% of the total maintenance cost for vehicles built in 1981, whereas PARTS comprise only 18% of the total maintenance cost for vehicles built in 1979.

In the table below, the total maintenance for vehicles in weight classes 1 through 4 are each compared to the total maintenance for vehicles in weight class 5. That is, TOTMAINT/%R5 displays the total maintenance for each set of the first four row sets (CLASS.1-4) as a percentage of the total maintenance for the fifth row set (CLASS.5).

```
AN> table class.*
VALUES: totmaint totmaint/%r5
                    TOTMAINT TO/%r5
CLASS.1 (8)       24,700.00    61%
CLASS.2 (8)       19,504.00    48%
CLASS.3 (8)       23,894.29    59%
CLASS.4 (8)       19,202.00    48%
CLASS.5 (8)       40,266.54   100%
```

In the next example, we compare the total maintenance cost for low mileage vehicles to the total maintenance cost for high mileage vehicles by weight class. The total maintenance for the second column set LOMILE (low mileage vehicles) is compared to the total maintenance for the first column set HIMILE (high mileage vehicles) as a percentage, when these two sets are intersected with the sets for the 5 weight classes. #INT is the number of items in the intersection of the row and column sets. This function is described in Section 12.10.3 “Int ersection Functions” below.

```
AN> table
ACROSS THE TOP: himile lomile
DOWN THE SIDE: class.*
VALUES: totmaint totmaint/%c1 #int

                    HIMILE (20)                    LOMILE (20)
              TOTMAINT    TO/%C1  #INT    TOTMAINT    TO/%C1  #INT
CLASS.1 (8)  12,210.00      100%     3   12,490.00      102%     5
CLASS.2 (8)  17,374.00      100%     4    2,130.00       12%     4
CLASS.3 (8)  17,019.53      100%     3    6,874.76       40%     5
CLASS.4 (8)  12,826.00      100%     4    6,376.00       50%     4
CLASS.5 (8)  31,423.00      100%     6    8,843.54       28%     2
```

In the example above, the total maintenance cost in the cell representing the intersection of sets LOMILE and CLASS.3 is $6,874, which is 40% of $17,019.53. That is, for the eight vehicles of weight class 3, the total maintenance cost for the 5 low mileage vehicles is 40% of the maintenance costs for the 3 high mileage vehicles in the same weight class.

12.10.3 Intersection Functions

As well as specifying field names to the "VALUES:" prompt, the TABLE command may be used to display information about the size (item count) of the intersection of the column (across the top) and the row (down the side) sets.

Function	Meaning
#INT	Size of the intersection, i.e. the number of items in the intersectio.
%TOT	#INT as a percentage of the total items in the logical master file.
%ROW	#INT as a percentage of the row set size.
%COL	#INT as a percentage of the column set size.
SIG	A measure of the statistical significance of the intersection size. A value (positive or negative) above .9000 indicates significance. SIG is determined using the Fisher Exact Test. (See Section 12.10.5 for a discussion of the Fisher Exact test.)
ALL	All of the above.

In the next two dimensional table, the statistical significance of the intersection is determined by the Fisher Exact Test (SIG).

```
AN> table
ACROSS THE TOP: old new
DOWN THE SIDE: himaint lomaint
VALUES: #int sig
                    OLD (24)          NEW (16)
                 #INT      SIG     #INT      SIG

HIMAINT (32)       24     .9998        8     -.999

LOMAINT (8)         0     -.999        8     .9998
```

This table tells us that vehicle age and vehicle maintenance costs are "very related" to each other. Hardly a surprise!

The following is a two dimensional table in which the item count in the intersection of the row sets and column sets is examined. In this example, the 4 items in the intersection of sets HIPARTS and HEAVY, represent 10% of total items (40), 15% of the row set HIPARTS (26) and 44% of the column set HEAVY (9).

```
AN>table
ACROSS THE TOP: light med heavy
DOWN THE SIDE: hipart himaint
VALUES: #int %tot %row %col
                LIGHT (4)              MED (18)              HEAVY (9)
            #INT %TOT %ROW %COL #INT %TOT %ROW %COL #INT %TOT %ROW %COL
HIPARTS(26)  1   2%   3%  25%   14  35%  53%  77%    4  10%  15%  44%
HIMAINT(32)  3   7%   9%  75%   12  30%  37%  66%    8  20%  25%  88%
```

12.10.4 Table Formats

The following diagram displays the various table formats.

```
-----------------------------------------------------------------------
|                    1. LISTING FOR THE ENTIRE FILE                   |
|                                                                     |
| field1 field2 field3 field4 ...                                     |
|  ...    ...    ...    ...       Data can be sorted in any order    |
|  ...    ...    ...    ...                                          |
| ______ ______ ______ ______                                        |
| total1 total2 total3 total4     (For T fields only, see            |
|                                        Section 12.13.1)             |
-----------------------------------------------------------------------
|                         2. LISTING FOR A SET                        |
|           Set 1                                                     |
| field1 field2 field3 field4 ...                                     |
|  ...    ...    ...     ...      Data can be sorted in any order    |
|  ...    ...    ...     ...                                         |
| ______ ______ ______ ______                                        |
| total1 total2 total3 total4     (For T fields only)                |
-----------------------------------------------------------------------
|                          3. VALUE SUMMARY                           |
|                     Horizontal Set by Set Format                    |
|       Set1                 Set2          etc.                       |
| fld1/op* fld2/op ..   fld1/op fld2/op ..                            |
|   ...      ...           ...     ...                                |
|   ...      ...           ...     ...                                |
-----------------------------------------------------------------------
|                          4. VALUE SUMMARY                           |
|                       Vertical Set by Set Format                    |
|       fld1/op*  fld2/op  fld3/op ..                                 |
| Set1    ...      ...     ...                                        |
| Set2    ...      ...     ...                                        |
| Set3    ...      ...     ...                                        |
| ...                                                                 |
-----------------------------------------------------------------------
|         5. TWO DIMENSIONAL CROSS TABULATION OF INTERSECTIONS         |
|                            Set x Set Format                         |
|           Col_Set1  Col_Set2  Col_Set3 ...                          |
| Row_Set1   ...**     ...       ...                                  |
| Row_Set2   ...       ...       ...                                  |
| Row_Set3   ...       ...       ...                                  |
-----------------------------------------------------------------------
|            6. TWO DIMENSIONAL CROSS TABULATION OF VALUES            |
|                            Set x Set Format                         |
|            Col_Set1          Col_Set2   ...     Total***           |
|          fld1/op* fld2/op  fld1/op fld2/op    fld1/op fld2/op       |
| Row_Set1   ...      ...      ...     ...        ...     ...        |
| Row_Set2   ...      ...      ...     ...        ...     ...        |
```

```
| Row_Set3   ...      ...      ...     ...        ...     ...        |
| Total***   ...      ...      ...     ...        ...     ...        |
-------------------------------------------------------------------------

*Operation (op) can be:
   o total value in set (/V)
   o average value in set (/A)
   o minimum value in set (/MI)
   o maximum value in set (/MA)
   o count of non-zero values in set (/E)
   o median of values in set (/ME)
   o standard deviation (population) (/SP)
   o standard deviation (sample) (/SD)
   o value in set as a percentage of the row sum (/%R)
   o value in set as a percentage of the column sum (/%C)
   o value as a percentage of same field in nth column set (/%Cn)
   o value as a percentage of same field in nth row set (/%Rn)
   o value as a percentage of the nth field in the same cell (/%n)

**The size of the intersection can be shown:
   o as itself  (#INT)
   o as a percentage of the column set size (%COL)
   o as a percentage of the row set size (%ROW)
   o as a percentage of the total file size (%TOT)
   o in terms of its statistical significance (SIG)
   o or any combination of these or all of these (ALL)

***Total is a pseudo-set name which is used to total the values
for each field in the column set or the row set or both.
```

12.10.5 The Fisher Exact Test

The intersection function SIG uses the Fisher Exact Test to measure the statistical significance of the intersection. The Fisher Exact Test (FET) is a measure of how the actual intersection size relates to an expected intersection size. The expected intersection size is determined by assuming the row and column attributes are each distributed randomly among the item population, i.e. that the row and column attributes are independent of each other. There is a probability associated with each logically possible intersection size. This probability distribution is a curved graph of what we might expect the intersection size to be.

The SIG value is the probability of obtaining the observed intersection size (or one more extreme) given the marginal totals of the two sets being intersected (assuming that there is no relationship between the attributes of the intersected sets). In other words, if the two sets being intersected are not related what is probability of obtaining the observed intersection value RANDOMLY. If that probability is low than SIG is high, meaning that the attributes for the two sets being intersected probably are related. A minus sign is displayed with SIG when the intersection size is less than one would expect randomly.

Hence, if the SIG is .9 or higher, this means that the actual intersection size fell at either the low or high ends of the curve. That is, it is very unlikely that the row and column attributes were independent of each other, and conversely, it is quite likely there is a relationship between them.

12.11 The GRAPH Command

The GRAPH command is a reporting tool for displaying summary values in tables and graphs for the items in the requested sets. The syntax and table formats for the GRAPH are similar to those of the TABLE command (see Section 12.10 "The TABLE Command"), except that the GRAPH command has several restrictions (see Section 12.11.2 "Graph Restrictions and Conventions")

Graphs can be produced on all DEC VT-compatible video terminals, or on standard ASCII hard copy terminals.

12.11.1 GRAPH Syntax

There are two variations of the GRAPH command syntax, namely the one dimensional and the two dimensional graphs. Below are the syntax options followed by a description and examples.

12.11.1.1 GRAPH on a List of Sets, One Dimensional

If set names are included on the GRAPH command line, then GRAPH assumes that these sets will be "down the side". GRAPH (clears the screen on video terminals) and immediately prompts for "VALUES:" to which the user enters the names of the fields to be displayed. Fields may include actual fields, linked fields or created fields. The word SAME may be entered on the GRAPH command line or to the "VALUES:" prompt, which instructs GRAPH to use the previously entered response to that prompt. The general syntax for creating a one dimensional graph follows.

```
AN> graph set-name1 [set-name2] [same]
VALUES: field1[/op] [+] [field2/op] [#int] [same]
```

By default, the values being graphed are subtotals for the fields requested. That is, the field operation "/V" is assumed. The field operations "/AV", "/MI", and "/MA" may be appended to the field name to display the average, minimum and maximum values for that field. These field operations are described in Section 12.10.2 "Operations On Values". Note that unlike the TABLE command which can display detail and/or summary values, the GRAPH command displays summary values **only.** Other restrictions which apply to GRAPHS are described in Section 12.11.2 "Graph Restrictions and Conventions".

In the following example the fields PARTS and LABOR are graphed for the sets built to three different model years (Y.81, Y.80, Y.79). The scale option (see Section 12.19.8 "Scale") is set to 2000.

```
AN> graph y.81 y.80 y.79
VALUES: parts labor

            PARTS    LABOR   <-----------------2000------------
------>

 Y.81 (4)  1,372.00 1,418.00 [==========================]

                             [XXXXXXXXXXXXXXXXXXXXXXXXXX]

 Y.80 (5)  1,563.00 1,688.00 [============================]

                             [XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX]

 Y.79 (5)  1,974.76 1,956.00
[======================================]

                        [XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX]
```

12.11.1.2 Two Dimensional Graph

In a two dimensional graph (i.e. cross tabulation) sets are placed both "across the top" and "down the side". The values displayed in the cells of the table and in the bar graphs represent the items in the **intersection** of the row and column sets. The values displayed can be totals, counts, averages, minimums, and maximums, based upon the items in the intersection of the row and column sets.

To create a two dimensional graph, the GRAPH command is followed by pressing ENTER. GRAPH prompts for the "ACROSS THE TOP:" and "DOWN THE SIDE:" sets. In each case the user enters one or more set names or SAME. If SAME is entered, GRAPH uses the previously entered response to that prompt. GRAPH then prompts for "VALUES:" and the user enters the field names to be displayed which may include actual fields, created fields and linked fields.

The Intersection Functions #INT, %COL, %ROW, %TOT and SIG may be displayed in the table and on the graphs, where applicable.

The general syntax for two dimensional graphs follows.

```
AN> graph
ACROSS THE TOP: set-name1 [set-name2 ...] [same]
DOWN THE SIDE: set-name1 [set-name2 ...] [same]
VALUES: field1[/op] [+] [field2/op ...] [int function] [same]

Intersection functions include #INT %TOT %COL %ROW SIG.
```

12.11.1.2.1 One Set Across the Top

If there is **one** set "across the top", then multiple fields can be graphed. Each field displays as a separate horizontal bar graph. Fields may be added together on a single bar graph, and displayed in alternating shades by placing a plus sign (+) between the fields (see Section 12.11.3 "Adding Values on the Same Bar Graph"). If a field name is surrounded by parentheses, the field is displayed in the table but not in the graph.

In the following example, the set of high maintenance vehicles (HIMAINT) displays across the top. The sets to high and low mileage vehicles (HIMILE and LOMILE) display down the side. The values are the average cost for parts and labor (PARTS/AV and LABOR/AV) for the items in the intersection of these sets. The scale option (see Section 12.19.8 "Scale") is set to 1000.

```
 AN> graph
ACROSS THE TOP: himaint
DOWN THE SIDE: himile lomile
VALUES: parts/av labor/av

                HIMAINT (32)
          PARTS/AV LABOR/AV <------------------1000------------
------>
 HIMILE      458.25   630.64 [=================]

                             [XXXXXXXXXXXXXXXXXXXXXXXX]

 LOMILE      345.37   427.06 [============]

                             [XXXXXXXXXXXXXXX]
```

12.11.1.2.2 Multiple Sets Across the Top

If there are **multiple** sets "across the top", then only **one** field can be graphed. The fields within a row of the table are graphed in alternating shades.

In the following example, the sets HILABOR (high labor cost vehicles) and HIMAINT (high maintenance cost vehicles) display "across the top". The sets beginning with "Y". (three model years: 81, 80 and 79) display "down the side". The values to be graphed are the average maintenance cost (TOTMAINT) for the items in the intersection of the column and row sets. The scale option (see Section 12.19.8 "Scale") is set to 6000.

```
AN> graph
ACROSS THE TOP: hilab himaint
DOWN THE SIDE: y.*
VALUES: totmaint/av

    HILAB (28) HIMAINT (32)
    TOTMAIN/AV TOTMAIN/AV<------------------6000------------
------->

 Y.81 (4)    893.00  1,055.00 [====|XXXXXX]

 Y.80 (5)  1,493.00  1,790.00 [========|XXXXXXXXXXX]

 Y.79 (5)  3,028.92  2,540.94
[==================|XXXXXXXXXXXXXXXX]
```

12.11.2 Graph Restrictions and Conventions

There are several restrictions and conventions which apply to graphs but not to tables.

1. The fields to be graphed must be numeric.
2. Fields may be displayed in the table but **not** in the graph by enclosing the field in parentheses. (e.g. (MODYR))
3. Only summary values can be graphed. Detail values from each item in the set or from the whole file may not be graphed.
4. The only values operations which may be graphed are: "/V" (subtotal), "/AV" (average), "/MI" (minimum), and "/MA" (maximum). Other field value operations may be included in the table but not on the graph by enclosing the field in parentheses.
5. If there are multiple sets across the top, then only **one** field may be graphed.
6. If multiple fields are being graphed for each set, the bar graphs from top to bottom represent the fields from left to right.

12.11.3 Adding Values on the Same Bar Graph

As we saw in Section 12.11.1.2.2 "Multiple Sets Across the Top" above, if there are multiple sets across the top, each row is graphed with a single horizontal bar, in which the fields are displayed in alternating shades.

GRAPH can also be instructed to display several **different** fields on the same horizontal bar graph. If the fields entered to the "VALUES:" prompt are separated by a plus sign (+), GRAPH adds these fields together creating a horizontal bar graph where the fields are graphed in different shades in the same horizontal bar. We call this type of graph an Additive Horizontal Bar Graph.

In the following example, the fields PARTS and LABOR are added on the same horizontal bar graph for each of the sets to the model years 81, 80 and 79. Note that the scale option is set to zero for the graphs below.

```
AN> graph y.81 y.80 y.79
VALUES: parts + labor

                PARTS      LABOR
 Y.81 (4)     1,372.00   1,418.00 [=============|XXXXXXXXXXXXX]

 Y.80 (5)     1,563.00   1,688.00 [===============|XXXXXXXXXXXXXXXX]

 Y.79 (5)     1,974.76   1,956.00
[==================|XXXXXXXXXXXXXXXXXX]
```

12.11.4 Graphing Negative Values

The GRAPH command may be used to graph both positive and negative values. If the values to be graphed are negative, the zero point of the graph is the midpoint of the graph. Positive values extend to the right and negative values extend to the left of the midpoint.

For example, in the budget analysis, the field DIFF is created using the CREATE command (see Section 12.16 "The CREATE Command") to represent the difference between a budget and actual amount. The budget and actual amounts are displayed on the table but are not graphed, by enclosing these fields names in parentheses. The field DIFF is graphed as follows. The scale option (see Section 12.19.8 "Scale") is set to 22000.

```
AN> graph dept*
VALUES: (budget) (actual) diff

            BUDGET    ACTUAL      DIFF 11000----------
++++++++++++11000
 DEPT1(43) 17,491.64 15,890.16  1,601.48                [=]
 DEPT2(49) 17,087.84 19,706.88 -2,619.04             [==]
 DEPT3(62) 22,551.54 17,329.68  5,221.86                [======]
 DEPT4(204)64,915.29 75,485.52-10,570.23 [==============]
```

12.11.5 Summary of Graphs Types

In summary there are four types of graphs.

1. **Positive Horizontal Bar Graph**

   The positive horizontal bar graph displays individual fields with individual bars. The origin of these graphs is the left-most portion of the graph section of the screen.

2. **Additive Horizontal Bar Graph**

   The additive horizontal bar graph displays two or more fields within a single horizontal bar. The size of the section for each field represented as a shade is based on the value of that field, and the scale option setting, if any. The fields on a additive graph are in the same left to right order as they are in the data display.

3. **Signed Horizontal Bar Graphs**

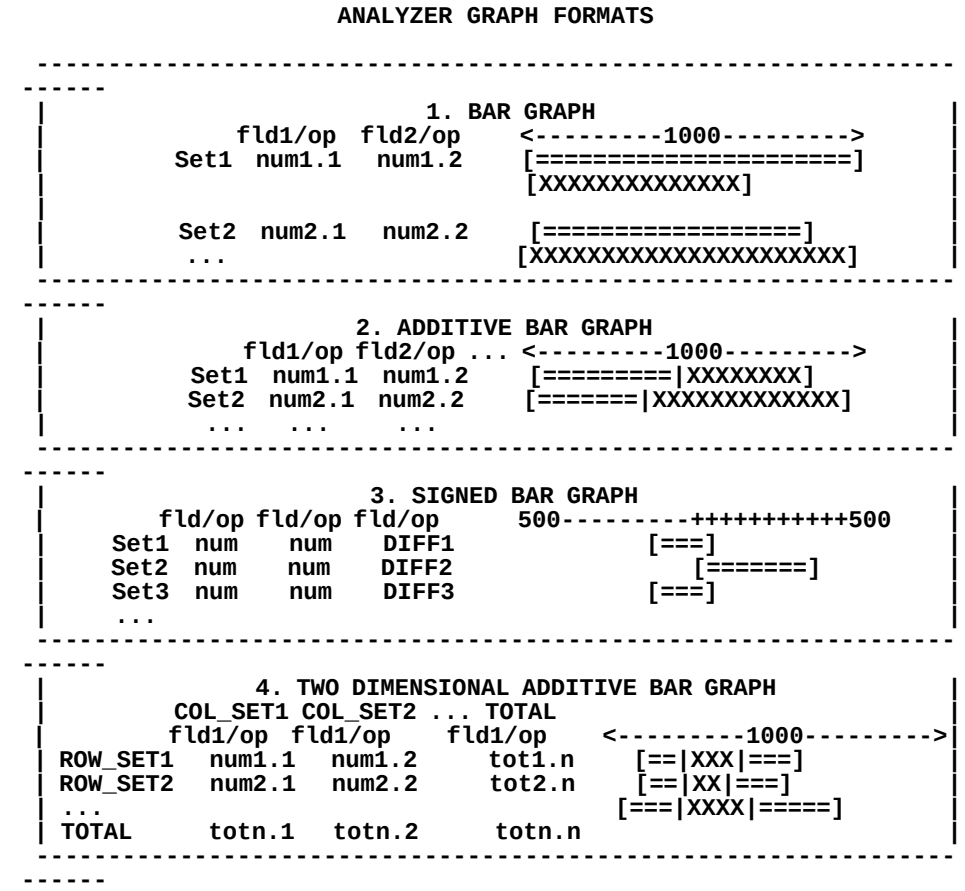
   The signed horizontal bar graph is used to display negative and positive values. Typically, this graph displays differences or variance between estimated (or expected) and actual values (e.g. actuals over or under budget). In this type of graph the zero point is the midpoint of the graph. Graph sections for positive values extend to the right, and graph sections for negative values extend to the left of the midpoint.

4. **Multi-Set Horizontal Bar Graph**

   The multi-set horizontal bar graph displays how the "across the top" sets divide up a value (or simply the intersection size) for a list of "down the side" sets. For example, for the different weight classes of vehicles ("down the side"), how does the total maintenance cost (the "value") differ across model years ("across the top").

   This graph is similar to the "Additive Horizontal Bar Graph" except this graph "adds" the same fields for different "across the top" sets, whereas the Additive Horizontal Bar Graph adds different fields for the same "across the top" set (or the entire logical file).

The following diagram shows the different graph types.



12.11.6 Scale Option

There are two factors which influence the size of a particular graph. The maximum size of the graph is determined by the amount of space between the right most column of the table and the page width (see the OPTION command Section 12.19 "The OPTION Command"). The scale option (see Section 12.19.8 "Scale") also affects the size of the horizontal bar graph.

If the scale option is set to zero (the default), the scale of the graph is determined from the values of the fields being graphed. For each different field being graphed, the length of the graph is relative to the highest value for that field.

If the scale option is non-zero, then the width of each unit in the graph is based on the value of the scale option.

If the scale option is -1, the entire display width is used for the highest value being graphed, all other values in all fields are graphed relative to that highest value.

The types of graph outlined above may be used with the scale option either set to zero or to a non-zero value. Note that if the scale option is non-zero, then that scale value applies to **all** values being graphed. If the SCALE option is zero, i.e. graph scaling is derived directly from the data values, then **each** field in the graph determines the scaling only for the graphs for that field.

12.11.7 Graph Shadings

Each graphed field is assigned a shading. The shadings are graphed in the order described below.

TERMINAL	SHADING
HARD COPY	1) open bar 2) filled bar (uses XXX)
VT (no advance video)	1) reverse video, 2) ### (pound sign)
VT (advanced video)	1) reverse video, 2) bright reverse video

12.12 The SHOW Command

The SHOW command is an "informational" command to allow the user to review data names and operations. The SHOW command can display either all fields, all labeled fields, all created fields, all sets, all labeled sets, or particular sets or fields. SHOW can also display the status of various printing and reporting options. In addition, SHOW can display the log of the commands in the current "SAV" file. SHOW may also be used to display the fields in another ADMINS data file which may be named as the master file or linked into the logical master file.

12.12.1 Show Fields

Field names are displayed in the order in which they were "defined" in the original master file. Following the fields from the master file are the created and linked fields, if any.

12.12.1.1 Show All Fields, in Detail

To display a list of all fields, the number of characters in the print-width, the ADMINS field data type, the totaling status, the scaling factor,[7] and the name of the link file (in the case of linked fields) the user types the "SHOW FIELDS" command. SHOW FIELDS may be abbreviated to "SHOW F".

```
AN> show fields
VEH#          4   NT  -  X9999
MODYR         6   NT  -  I
TOTMILES     12   T   -  D
MILES81      12   T   -  D
WTCLASS       6   NT  -  I
TOTMAINT     12   T   -  D2
PARTS        12   T   -  D2
LABOR        12   T   -  D2
YRACQ         6   NT  -  I            REGISTER.MAS
YRSERV        6   NT  -  I            Created Field
AGE           6   NT  -  I            Created Field
```

12.12.1.2 Show Fields, Briefly

A compact list of the fields in the logical master file is displayed by typing "SHOW BF":

```
AN> show bf
VEH# MODYR TOTMILES MILES81 WTCLASS TOTMAINT PARTS LABOR
ANSEQ YRACQ YRSERV AGE
```

7. Totaling status and scaling factor are described in Section 12.13.1 "Using the NAME Command with Fields".

12.12.1.3 Show All Created Fields

A list of all the created fields with their derivation rule is displayed by typing "SHOW CRF". Created fields are described in Section 12.16 “The CREATE Command”.

```
AN> show crf
   YRSERV       6   NT  -  I            Created Field
         CREATE YRSERV/I 82 - YRACQ
   AGE          6   NT  -  I            Created Field
         CREATE AGE/I 82 - MODYR
```

12.12.1.4 Show a Particular Field

To see a field (or several fields) with the details described in SHOW Fields type:

```
AN> show field-name1 field-name2 ...
```

12.12.1.5 Show All Labeled Fields

A list of all the fields which have a descriptive label is displayed by typing "SHOW LF":

```
AN> show lf
```

12.12.2 Show Sets

When a set is "shown", the item count, the sequential set number, the set operations used to build the set, and the set label (if any) is also displayed. Sets are displayed in reverse sequence order (i.e. the newest ones first).

12.12.2.1 Show All Sets

To display all the sets, type "SHOW SETS". SHOW SETS may be abbreviated to "SHOW S".

```
AN> show sets
.
.
.
4       Y.80          5      MODYR EQ 80
3       Y.81          6      MODYR EQ 81
2       HILAB         27     LABOR81 GE 300.00
1       HIPARTS       26     PARTS81 GT 300.00
```

12.12.2.2 Show All Labeled Sets

The user types the following command to display a list of only the sets which have a descriptive label.

```
AN> show ls
```

12.12.2.3 Show the Last "n" Sets

To display only the last "n" sets which were built, type "show -n". For example, to display the last "5" sets type:

```
AN> show -5
18      NEWXHEAVY    16    XO NEW HEAVY
17      LOMILIT       5    CO HIMILE LIGHT
16      HIMAINT      37    CO LOMAINT
15      LOMILE       30    CO HIMILE
14      NEW          14    UN Y.81 Y.80 Y.79  Cars built since 79
```

12.12.2.4 Show a Particular Set

To display one specific set or several sets type:

```
AN> show s:set-name1 [s:set-name2 ...]
```

The "S:" preceding the set name is only required if there is a field name which matches the set name. Otherwise the "S:" is optional.

Particular sets may also be displayed by entering the set sequence number. In this case the "S:" is required before the set number. For example, to display sets numbered 5 and 8 type:

```
AN> show s:5 s:8
```

12.12.3 Show File-Name

The SHOW file-name command displays a list of the fields from other ADMINS data files. This feature may only be used with data files with the MAS or TAB file type. For example, to see the data fields in the file PREV.MAS, type:

```
AN> show prev.mas
VEH# MODYR TOTMILES MILES81 WTCLASS TOTMAINT PARTS LABOR
```

12.12.4 Show Options, Show Command Log

The current ANALYZER options settings (see Section 12.19 “The OPTION Command” for a discussion of the various ANALYZER options) can be displayed by typing:

```
AN> show op
```

The ANALYZER continuously logs all of the set building and definitional commands. The contents of this log may be viewed by typing:

```
AN> show com
```

The contents of the command log is further explained in Section 12.18 “The WRITE Command”.

The SHOW command is also used to display the name of the active SAV file (see Section 12.2 “The FILE Command and SAV Files”).

```
AN> show sav
```

12.13 The NAME Command

The NAME command is used to rename and/or change field names, labels or field characteristics, set names or set labels.

12.13.1 Using the NAME Command with Fields

The NAME command is used to rename a field and/or assign or change the field label or any of the field characteristics.

```
AN>name field [f:new-fldname][prt-width[/prt-
len]][tnt][s.d][label]
```

When renaming a field, the new field name must be preceded by "F:". For example, to change the field name PARTS81 to the name PARTS, type:

```
AN> name parts81 f:parts
```

NAME can also be used to change the number of characters in the "print width" of a field for use in tables, or to change the print width and length of TXnn or TInn fields. Default print widths are derived from the ADMINS field type (e.g. integer=7, decimal=12, pictured=as specified, TI60=60, TX76=76, etc.). **The length qualifier applies only to text fields.** The default print length for a text field is 0, which means **there is no constraint on the number of lines that may be printed** from the text field. If the length qualifier is set to a negative number in the NAME command, the field is treated as an alphanumeric field (i.e. only the first line is displayed).

In the following examples, the print width of the field TOTMAINT is changed to 8 characters, and the print width and length of the text field COMMENTS (a TX70 field) are changed to 60 and 40, respectively.

```
AN> name totmaint 8

AN> name comments 60/40
```

NAME can be used to change the totaling status of numeric fields. By default, the ANALYZER designates all decimal fields (Dn) and four word decimal fields (Fn) as "T" for automatically totaling and all other fields as "NT" for non-totaling. In tables which display detail values, the total value is also displayed for any fields with a totaling status of "T".

In the following example the totaling status of the field LABOR is changed from automatically totaling to non-totaling.

```
AN> name labor nt
```

NAME is also used to set a scaling factor for decimal fields. The scaling factor has two components, the scale and the decimal point. The number is divided (and rounded) by the power of ten corresponding to the scale, and then the new value is displayed with the new decimal point setting.

For example if the field BALANCE has values in the millions (e.g. 1,242,544) which the user would prefer to express as "1.24", the scale would be set to "6.2" for this field. This tells the ANALYZER to divide the values in the field by 1,000,000 (ten to the **sixth** power), and then express the result with **two** decimal places. This command follows:

```
AN> name balance 6.2
```

To remove the scaling factor, the user sets the scaling factor to "0.0". For example:

```
AN> name balance 0.0
```

12.13.2 Using the NAME Command With Sets

The NAME command is also used with sets to rename a set and/or assign or change a set label.

```
AN> name s:set-name [s:new-set-name] [label]
```

Examples of the use of the NAME command with sets follow. In the first example NAME is used to change the set name. Notice that the "S:" is required before each set name. In the second example, NAME is used to assign (or change) a set label. In the third example, NAME is used to remove a set label.

```
AN> name s:himile s:himileage

AN> name s:hiparts parts gt $300.

AN> name s:hiparts
```

To rename a group of sets that begin with the same string of characters use the asterisk (*) as a wildcard character:

```
AN> name s:state.* s:xstate.*
SET S:STATE.1 RENAMED TO S:XSTATE.1
SET S:STATE.2 RENAMED TO S:XSTATE.2
SET S:STATE.3 RENAMED TO S:XSTATE.3
SET S:STATE.4 RENAMED TO S:XSTATE.4
SET S:STATE.5 RENAMED TO S:XSTATE.5
```

12.14 The EXAMINE Command

The EXAMINE command is used to show the full breakdown of a set, i.e. the step by step construction used to build the set. This breakdown is displayed in a hierarchical fashion. The syntax for the EXAMINE command is:

```
AN> examine set-name
```

For example in a vehicle maintenance application, individual vehicles are scheduled for extensive preventive maintenance based on criteria derived from analysis of past experience. The set of vehicles (S:PREV) was built as follows:

1. All vehicles with total mileage over 50,000.
2. Small vehicles built before 1973.
3. Medium and large vehicles built before 1977.
4. Vehicles relatively inactive in 1981 (5000 miles or less in 1981) that required at least $300 maintenance in labor or parts.

The EXAMINE command would show this as follows:

```
    AN> examine prev

FIELD    INSTRUCTION              SET NAME       COUNT DESCRIPTION
-------- -----------              --- ----       ----- -----------
TOTMILES GT 50,000                    HIMILE          20
MODYR    LE 72                          YR72-           4
CLASS    EQ 1 2                         CL_1.2         16
         IN YR72- CL_1.2              OLDSMALL          1
MODYR    LE 76                          YR76-          18
CLASS    EQ 3 4 5                       CL_3.4.5       24
         IN YR76- CL_3.4.5            OLDBIG           11
PARTS    GT 300.00                        HIPARTS      26
LABOR    GE 300.00                        HILAB        28
         UN HIPARTS HILAB               TOT300$        33
MILES    LE 5,000                       INACTIV          3
         IN TOT300$ INACTIV           HI/INACT           1
         UN HIMILE OLDSMALL OLDBIG PREV                 23
            HI/INACT
```

In the EXAMINE command, the first column, FIELD, indicates the field which was used in building the set (e.g. TOTMILES, MODYR...). The INSTRUCTION column displays the selection criteria or the set operation which was used to build the set.

The SET NAME column shows the most current name for each set. The indentation shows the structure of the set building. Set names with the same left margin justification were used to build the set beneath them with the margin justification two spaces to the left. For example, the sets YR72- AND CL_1.2 were used to build the set OLDSMALL. Also, the sets HIMILE, OLDSMALL, OLDBIG, and HI/INACT were used to build the set PREV.

The COUNT column shows the item count for each set, that is, the number of records. The DESCRIPTION column displays the set labels, if any.

12.15 The HELP Command

The HELP command provides on-line assistance and explanations of the ANALYZER commands and their variations.

To display the command list, the user types "HELP" to the "AN>" prompt. For a more detailed explanation of a specific command the user types the HELP command followed by at least the first two letters of the command in question. Additional key words documented are listed at the end of the command explanations. To display the additional information the user types (at least) the first two letters of the command followed by the first two letters of the keyword.

```
AN> help [command] [keyword]
```

For example to display documentation on the CREATE command, enter the following:

```
AN> help create
```

12.15.1 Location of the HELP File

The ANALYZER HELP file, ANALYZ.HLP, should be located on the disk and directory assigned to the logical name ADM$DIST.

12.16 The CREATE Command

The CREATE command is used to define a new virtual field. The CREATE command in the ANALYZER uses the same syntax as the CREATE statement in REPORT (see Section 7.13.1 "CREATE Statement" for a description of the CREATE statement, and see Chapter 8: "Expressions" for a description of ADMINS expressions and syntax). Once created, the new virtual field is part of the logical master file and can be used as if it were an actual field in the master file.

CREATE does not perform any i/o. It simply sets up a definition which is performed each time the new virtual field is needed (e.g. when building a set based on the values in the CREATEd field, or when displaying the CREATEd field in a table).

All links are performed **before** the CREATEd fields are evaluated. Therefore, links cannot be performed using CREATEd fields as the key. See Section 12.3 "The LINK Command" on the LINK command.

ANALYZER CREATE automatically manages the decimal point in arithmetic operations between decimal fields of the same or different decimal point settings, between decimal fields and constants, and between more than one constant. (This is unlike other ADMINS commands in which automatic decimal point management is available as an option but is not the default.)

The following example illustrates both the use of CREATE and also automatic decimal point management by the ANALYZER.

In this example file EXAMPLE.MAS has the following fields:

Field	Type
D2	D2
D	D

and the following values:

```
D2          D
100.00      1,000
200.00      2,000
300.00      3,000
400.00      4,000
500.00      5,000
600.00       6,000
```

The CREATE command is used to create several virtual fields as described below.

In the following example, a field is multiplied by a constant, with the result field the same decimal point setting as the original field (D2).

```
AN> create 1fld/d2 d2 * 10
```

In the next example, a field is multiplied by a constant with the result field having a different decimal point setting from the original field.

```
AN> create 2fld/d   d2 * 10
```

In the next example, two fields are multiplied, and the resulting decimal point setting is different from either of the original fields.

```
AN> create 3fld/d1  d * d2
```

In the next two examples, two fields are divided.

```
AN> create 4fld/d  d / d2

AN> create 5fld/d1 d2 / d
```

The TABLE below displays the values of the fields D2 and D, and the created fields which result from the definitions above.

```
AN> table d d2 1fld 2fld 3fld 4fld 5fld
    D       D2      1FLD   2FLD          3FLD   4FLD   5FLD

1,000   100.00  1,000.00  1,000     100,000.0     10     .1
2,000   200.00  2,000.00  2,000     400,000.0     10     .1
3,000   300.00  3,000.00  3,000     900,000.0     10     .1
4,000   400.00  4,000.00  4,000   1,600,000.0     10     .1
5,000   500.00  5,000.00  5,000   2,500,000.0     10     .1
6,000   600.00  6,000.00  6,000   3,600,000.0     10     .1
------  ------- --------- ------  ------------  -----  -----
21,000 2,100.00 21,000.00 21,000   9,100,000.0     60     .6
```

12.17 Output Files

The ANALYZER OUTPUT command outputs records containing information derived in an ANALYZER session to an external ADMINS data file. This output file might be used to reorganize and consolidate the analysis steps performed on a file; or to build a subset of the existing logical master file; or to consolidate the relationships (LINKs, for example) developed in the analysis session directly into a single physical file. Sets built during the session can be recoded into new fields in the output file.

12.17.1 OUTPUT Syntax

OUTPUT command syntax is as follows:

```
AN> output file-name [all/keys] [s:set-name]
FIELD: field-name [type]
CODE:  value s:set-name
```

If the specified data file does not exist, OUTPUT creates a file definition instruction file (a "DEF") for the file. (If the file-name which follows the OUTPUT command does not include a file type, e.g. MAS, then the file type is taken from the file being analyzed.) Use this DEF file to DEFINE a file to receive the records to be output, then call OUTPUT again to output the records.

OUTPUT processes all the records of the ANALYZER master file unless a set name is specified on the OUTPUT command line. If a set name is given, only the records in that set are processed.

OUTPUT will place all the fields in the ANALYZER's virtual record in the output file if the keyword "ALL" is specified on the OUTPUT command line. The key fields of the main file are always included and will be the key fields of the output file unless KEYS is specified in the OUTPUT command line (see Section 12.17.3 "Output Files with New Key Structure"). Otherwise only those fields named in response to the FIELD: prompts are processed. If the names to be placed in the output file are present in the logical master file, the user does not provide a field type in response to the "FIELD:" prompt. The field type is used only when a new field is being created into which OUTPUT will recode sets.

When a field name is followed by a field type, the ANALYZER requests information on how it is to recode sets into values for the new field. First, it prompts "CODE:" on the next line. Enter a value followed by the set-name which is to be recoded as this value. Responding to the "CODE:" prompt with just carriage return by itself tells the ANALYZER that you are finished entering recode values for the field. (The ANALYZER prompts for another "FIELD:".)

Responding to the "FIELD:" prompt with just carriage return by itself tells the ANALYZER that you are finished entering fields.

12.17.2 Building Output Files

OUTPUT creates a DEF instruction file if the specified data file does not exist. After the file is created using the DEF file, call OUTPUT again to actually output the records.

Use the SPAWN* command to DEFINE the ADMINS data file without leaving the ANALYZER session:

```
AN> OUTPUT SAMPLE.MAS
FIELD:AFLD
FIELD:BFLD
FIELD:
SAMPLE.DEF CREATED
AN> SPAWN DEFINE SAMPLE

DEFSZ: 28 NF: 3  KEYLEN:1  RECSZ: 5  NRECS: 1000
  # OF BLOCKS   DATA: 16  INDEX: 8  TOTAL: 24
SAMPLE.DEF.1 CREATED
INDEXED file. KEYS are: N
AN> OUTPUT SAMPLE.MAS
SAMPLE.MAS OPENED
FIELD:AFLD
FIELD:BFLD
FIELD:
SAMPLE.MAS CLOSED 143 RECORD WRITTEN
```

The number of records written to the output file is determined by the size of the logical master file, or the size of the subset of the logical master file being written as output.

If the output file already has records in it, OUTPUT can be used to append records to the end of the file. In this circumstance the ANALYZER advises that there are already records in the file and awaits confirmation that processing should continue.

12.17.3 Output Files with New Key Structure

The OUTPUT command line qualifier, KEYS, provides the capability to create an output file with an entirely different key structure than the ANALYZER main file:

```
AN> OUTPUT file_name KEYS [s:set_name]
KEY:   any_field_name
FIELD: field_name [type]
 CODE:  value s:set_name
```

When KEYS[8] is specified the ANALYZER prompts with "KEY:". The user enters any fields (from the logical file, created or linked). Up to 9 key fields may be specified. Enter a carriage return by itself at the "KEY:" prompt to indicate that you are finished entering key fields. (The ANALYZER will prompt "FIELD:")

8. KEYS and ALL can not be on the same output line.

12.17.4 Example of the OUTPUT command

The following example illustrates the OUTPUT command syntax and the form of a file definition written by OUTPUT based on the data in the Motor Vehicle Preventive Maintenance file PREV.MAS.

Before using the OUTPUT command, an analysis environment is created by building sets, defining a link and a created field. The ANALYZER commands which were used prior to running the OUTPUT command are displayed below. Then the OUTPUT command instructions are run from a command file, EXAMPLE.COM.

```
$ an
ANALYZB1.SAV CREATED
AN> file prev.mas
AN> select
SELECT> class eq 5 s:large 6 passenger + vehicle
SELECT> class eq 3 4 s:midsize 5-6 passenger vehicle
SELECT> class eq 2 s:compact 4-5 passenger vehicle
SELECT> class eq 1 s:subcompact 4 passenger (or smaller)   vehicle
SELECT> cr
AN> link yracq from register.mas key is veh#
AN> create yrserv/i 82 - yracq
```

At this point, the sets and fields which we need for our output file have been defined. The command file which will be executed to build the output file has already been written. To call the command file EXAMPLE.COM we type @EXAMPLE to the "AN>" prompt. The DEF instruction file COST.DEF is created the first time the command file, EXAMPLE.COM is executed.

```
 AN> @example
 AN> output cost
 * The following is a list of the fields to be included in COST.DEF.
 FIELD: parts
 FIELD: labor
 FIELD: totmaint
 FIELD: yracq
 FIELD: yrserv
 FIELD: class
 * The field SIZE, is a new field derived from sets.
 * SIZE is an A6 field.
 FIELD: size a6
 * The field values and set-names to be coded into the field SIZE.
 * We assign "large" as the value for the items from the set
s:large.
 CODE: large  s:large
 CODE: mid    s:midsize
 CODE: cmpt   s:compact
 CODE: sbcmpt s:subcompact
 * To exit from the CODE prompt, a return is simulated with "cr".
 CODE: cr
 * To exit from the FIELD prompt, use "cr".
 FIELD: cr
 COST.DEF CREATED
 AN>
```

The following DEF instruction file, COST.DEF, was created by the OUTPUT command above.

```
COST.DEF
MAS 40
VEH# X9999 KEY1
PARTS D2
LABOR D2
TOTMAINT D2
YRACQ I
YRSERV I
CLASS I
SIZE A6
*    LARGE LARGE     6 passenger + vehicles
*    MID MIDSIZE     5-6 passenger vehicles
*    CMPT COMPACT      4-5 passenger vehicles
*    SBCMPT SUBCOMPACT     4 passenger (or smaller) vehicles
```

Notice that the field SIZE is created by recoding the sets LARGE, MIDSIZE, COMPACT, and SUBCOMPACT into field values. The comments below the field SIZE are written by the OUTPUT command and show the field value, the set-name recoded into that value and the set label (if any) for each field value.

After the OUTPUT command has written a file DEF, the SPAWN command is used to DEFINE the master file.

```
AN> spawn define cost
```

Again the command file EXAMPLE.COM is called. This time the records will be appended to the file COST.MAS.

```
AN> @example
COST.MAS OPENED
FIELD: parts
FIELD: labor
FIELD: totmaint
FIELD: yracq
FIELD: yrserv
FIELD: class
FIELD: size a6
CODE: large  s:large
CODE: mid    s:midsize
CODE: cmpt   s:compact
CODE: sbcmpt s:subcompact
CODE: cr
FIELD: cr
COST.MAS CLOSED 40 RECORDS WRITTEN
AN> stop
```

To examine the data in COST.MAS, the user can use the ANALYZER, or other ADMINS tools such as TRANS. The ANALYZER TABLE command is used to display the data in COST.MAS in the following example:

```
$ an cost.mas
AN> TABLE VEH PAR LAB CLASS SIZE
VEH#       PARTS         LABOR    CLASS   SIZE
0001      132.00        231.50        2   CMPT
0002      543.00        654.00        3   MID
0003      435.60        546.32        4   MID
0004      546.43        674.04        1   SBCMPT
0005      100.00        200.00        2   CMPT
0006      345.76        897.34        5   LARGE
  .         .             .          .    .
  .         .             .          .    .
  .         .             .          .    .
```

12.17.5 Recoding Multi-Value Fields

As explained above, OUTPUT can be used to recode sets into a new field (e.g. the field SIZE) so that the values in the new fields correspond to sets created in the analysis session.

The OUTPUT command can also be used to recode multiple values into a single field. (The use of multi-value data is described in Section 12.22 "Multi-Value Data".) This happens when each of a group of sets is recoded into a value, and an item (record) can be present in **more than one** of these sets. The new field created by recoding multi-value data into one field must be an alphanumeric field. Multiple values are concatenated in the field separated by a space.

The ANALYZER may be used to reanalyze data recoded in this way by using the "character string search" operator, IN. Use of the IN operator is described in Section 12.4.1 "Single Set Syntax".

The following example shows the use of the OUTPUT command with multi-value fields. The same OUTPUT syntax is used for recoding fields as described above. The sets which follow were built from a personnel file. They contain employees who speak a specific language. These sets will be recoded and incorporated into the new field called LANG in the output file EMPSKILL.

```
S:ENGLISH
S:FRENCH
S:GERMAN
S:RUSSIAN
S:CHINESE
S:DANISH
S:SWEDISH
S:SPANISH
S:ITALIAN
```

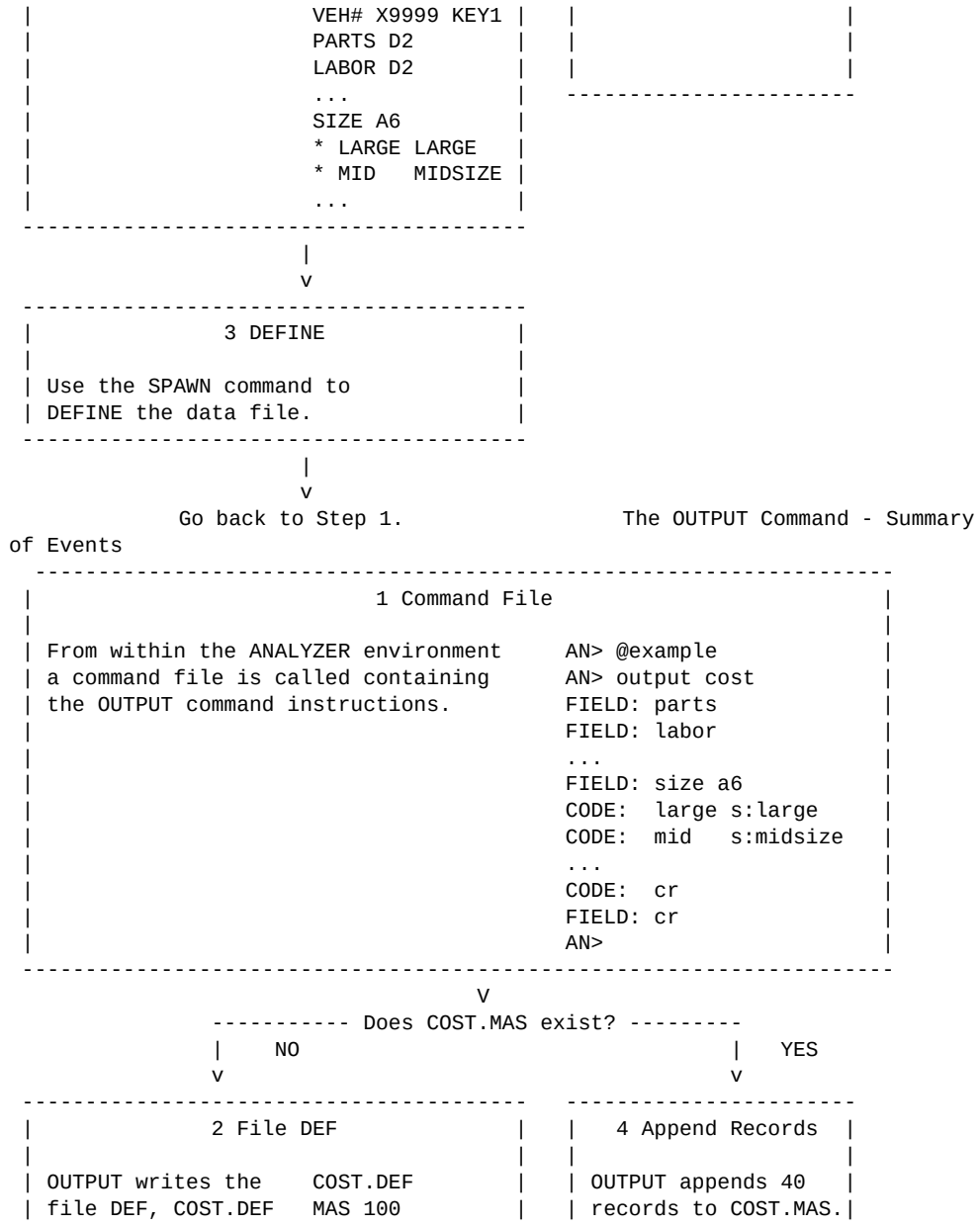
The following OUTPUT command file is used to create the file EMPSKILL (i.e. EMPSKILL.DEF and EMPSKILL.MAS). Since each employee may have skills in more than one language, the multiple values for language skills will be stored in the field LANG. The command file, LANG.COM is called from within the analysis session.

```
AN> @lang
AN> output lang
FIELD: typspd
FIELD: shthd
FIELD: wdproc
FIELD:  .
FIELD:  .
FIELD:  .
FIELD: lang a30
CODE: en s:english
CODE: fr s:french
CODE: ge s:german
CODE: ru s:russian
CODE: ch s:chinese
CODE: da s:danish
CODE: sw s:swedish
CODE: sp s:spanish
CODE: it s:italian
CODE: cr
FIELD: cr
```

The field LANG is an A30 field. This will allow for up to ten values (two letters each) to be stored within the field with one space between values.

12.17.6 Output Files: Summary

The flow of information from the command file containing the OUTPUT command, until records are appended into a master file, is diagrammed on the following page based on the output command file example in Section 12.17.4 “Example of the OUTPUT command”.



12.18 The WRITE Command

The WRITE command is used to create a text editable ANALYZER command instruction file from the internal log of commands in the active SAV file. The commands written into the command file include all of the set building, option setting and definitional instructions. Therefore, the command file created via WRITE can be used to re-create all the analysis steps up to the current point. Display commands, such as TABLE, SHOW, EXAMINE and OUTPUT are not written in the command file.

12.18.1 WRITE Syntax

The syntax for the WRITE command is as follows.

```
AN> write file-name
```

The WRITE command creates a text editable ANALYZER command instruction file containing all of the set building, option setting and definitional commands used in the analysis. The name of the command file created by the WRITE command is file-name.COM. The user does not include the file type on the WRITE command line because the ANALYZER adds the file type.

After the command file is written, it can be edited and consolidated using a text editor.

Command files based on the internal log are further discussed in Section 12.21 “Command Files”, and in Section 12.2.2 “The SAV File”.

12.18.2 Example using the WRITE Command

The following example is an excerpt from an analysis session in which the WRITE command is used. First level sets are built using the SELECT command. Additional sets are created using the UNION and COMPLEMENT commands. A TABLE is run on the data. Several options are set, the display width of fields is changed, a LINK is made to the file REGISTER.MAS. The log of the analysis is displayed using SHOW COM. Finally, the command file HICOST.COM is created using the WRITE command.

```
$ an
ANALYZB3.SAV CREATED
AN> file prev.mas
PREV.MAS 40 ITEMS, IS NOW ACTIVE
AN> select
SELECT> parts gt 400 s:hipart
SELECT> labor gt 400 s:hilab
SELECT> modyr eq 81 s:y.81
SELECT> modyr eq 80 s:y.80
SELECT> modyr eq 79 s:y.79
SELECT> totmile gt 50000 s:himile
SELECT> cr
...
AN> union y.81 y.80 y.79 = new
7     S:NEW        10    25%
AN> complement new = old
8     S:OLD        30    75%
AN> table
ACROSS THE TOP: new old
DOWN THE SIDE: y.*
VALUES: parts "/%r "/%c
...
AN> name parts 8
AN> name lab 8
AN> create diff/d parts - labor
AN> table y.*
VALUES: parts labor diff
...
AN> link all from register.mas key is veh#
AN> show com
1.    FILE PREV.MAS
2.    SELECT
3.    PARTS GT 400 S:HIPART
4.    LABOR GT 400 S:HILAB
5.    MODYR EQ 81 S:Y.81
6.    MODYR EQ 80 S:Y.80
7.    MODYR EQ 79 S:Y.79
8.    TOTMILE GE 50000 S:HIMILE
9.
10.   UNION Y.81 Y.80 Y.79 = NEW
11.   COMPLEMENT NEW = OLD
12.   NAME PARTS 8
13.   NAME LABOR 8
14.   CREATE DIFF/D PARTS - LABOR
15.   LINK ALL FROM REGISTER.MAS KEY IS VEH#
AN> write hicost
HICOST.COM WRITTEN 16 LINES
AN> stop
```

After exiting from the ANALYZER, it is possible to examine and modify the command file using a text editor. The contents of HICOST.COM are as follows:

```
FILE PREV.MAS
SELECT
PARTS GT 400 S:HIPART
LABOR GT 400 S:HILAB
MODYR EQ 81 S:Y.81
MODYR EQ 80 S:Y.80
MODYR EQ 79 S:Y.79
TOTMILE GT 50000 S:HIMILE
CR
UNION Y.81 Y.80 Y.79 = NEW
COMPLEMENT NEW = OLD
NAME PARTS 8
NAME LABOR 8
CREATE DIFF/D PARTS - LABOR
LINK ALL FROM REGISTER.MAS KEY IS VEH#
```

After editing (if necessary) this command file, HICOST.COM can then be used to re-create the analysis environment as follows:

```
$ an @hicost
```

12.19 The OPTION Command

The OPTION command allows the user to set various options for formatting and printing tables and graphs. To display the options list and current settings, type OPTION (or SHOW OP). The following is the OPTIONS list with default settings:

CODE	OPTION	MIN	MAX	DEFAULT
WI	page WIdth	20	254	80
LE	page LEngth	10	100	60
FO	FOrmat	0	2	0
RL	Row Label	6	40	10
CL	Column Label	6	40	20
PA	PAuse	0	1	1
LP	Line Printer	0	2	0
SC	SCale	-1	1000000	0
GR	GRoup size	0	6	0
SP	SPooler number	0	255	0
NC	Number of Copies	1	100	1

To change any of the default option settings, the two letter option code is followed by an equal sign (=) and the new value.

```
AN> option code=value
```

Each of the options is described in detail below.

12.19.1 Page Width

The page width (WI) option sets the maximum number of characters per line. The following example sets the page width to 132 characters.

```
AN> option wi=132
```

12.19.2 Page Length

The page length (LE) option sets the maximum number of lines per page for tables. Column headings are printed at the top of each new page. The following example sets the page length to 30 lines.

```
AN> option le=30
```

12.19.3 Format Control - Form Feeds

The format (FO) option controls the insertion of form feeds between pages produced by the TABLE command. There are three format control settings:

1. FO=0 indicates that the automatic form feed function is off. That is, form feeds are not issued at the end of pages.
2. FO=1 inserts form feeds in reports after each page, as determined by the page length value.
3. FO=2 issues one form feed immediately and then sets the format to 1, so that the report inserts form feeds between pages.

In the following example, format is set to 1.

```
AN> option fo=1
```

12.19.4 Row Label Width

Row label (RL) option sets the width for row set names and labels printed by the TABLE command. The default row label is ten characters, the minimum is six and the maximum is forty characters. To reset the row label width to 20 characters type:

```
AN> option rl=20
```

If there is insufficient space to display the row set name followed by the item count (which is enclosed in parentheses), the item count will be eliminated from the display and possibly the row set name or label will be truncated.

12.19.5 Column Label Width

The column label (CL) option sets the column width for each column set name and label "across the top". The column label option is used to accommodate large column set labels and to allow some flexibility in the spacing of the fields for the "across the top" sets. The column label spacing is determined as follows.

The TABLE command first determines if there is sufficient space within the current page width (see Section 12.19.1 “Page Width”) for the requested fields, an intervening space, and the "down the side" set names and labels (see Section 12.19.4 “Row Label Width”).

The print width of a particular field is a fixed amount. The default print width for a field is derived from the ADMINS field type unless the print width is reset with the ANALYZER NAME command, see Section 12.13.1 “Using the NAME Command with Fields”.

If there is sufficient space to display the requested fields, TABLE attempts to center the label (if any) for the column set(s) on the line directly below the column set name(s) and above the field names. The field names for the column sets are spread out as much as possible within the active column label setting. There is always at least one space between field names.

If the length of the column set label is greater than the column label setting, then the set label may be truncated.

In the following example, the column label is set to 25 characters.

```
AN> option cl=25
```

12.19.6 Pause

The pause (PA) option instructs the TABLE command whether or not to pause after printing one page of a table. If pause is set to 1 the TABLE command waits after each page (as defined by the page length option) prints, before printing the next page. The user presses ENTER to continue displaying the table, or types any character followed by ENTER to stop the report. If pause is set to 0 (zero), the complete table prints without any pauses. By default, pause is set to 1 when using the ANALYZER interactively, and pause is set to 0 when a table is run from an ANALYZER command file. In the following example, the pause option is set to 0.

```
AN> option pa=0
```

12.19.7 Line Printer

By default, the output from the TABLE command is displayed on the user's terminal. The line printer (LP) option can be used to redirect the table to a "LIS" file which can be either queued for printing, or sent immediately to another device. There are three possible settings for the line printer option.

1. LP=0 is the default. TABLE output is directed to the user's terminal (computer screen).
2. LP=1 directs the output to the terminal assigned to the logical name ADM$PRT0, if ADM$PRT0 is not assigned the output goes to a "LIS" file which is queued for printing to the queue assigned to the logical name ADM$SPOOLn[9] when the user exits from the ANALYZER.
3. LP=2 directs the output to the terminal assigned to the logical name ADM$PRT0, if ADM$PRT0 is not assigned the output goes to a "LIS" file which is queued immediately to ADM$SPOOLn without waiting for the user to exit from the ANALYZER.

Any table in a "LIS" file created by the ANALYZER is formatted as if FO=1, that is form feeds are automatically inserted between pages.

In the following example, the line printer option is set to 1.

```
AN> option lp=1
```

12.19.8 Scale

The scale (SC) option allows the user to set a scale size when using the GRAPH command as described in Section 12.11.6 “Scale Option”. The following example sets the scale size to 20,000.

```
an> option sc=20000
```

9. n is determined by OPTION SP, see Section 12.19.10 “Spooler Number”

12.19.9 Group

The group (GR) option instructs the TABLE and GRAPH commands to separate the "down the side" sets into groups. The number of sets in each group may range from 1 to 6. Each group is separated by an extra blank line in the table or graph. In graphs, groups of sets are assigned different shadings. The default setting for the group option is 0, which essentially has no effect.

In the following example, the group size is set to 4. That is, after each group of 4 "down the side" sets displays, a blank line separates that group from the next group of 4 sets.

```
AN> option gr=4
```

12.19.10 Spooler Number

The Spooler Number option (SP) is used to identify the logical name to be checked for queuing TABLE output. For example, if SP=6 the ANALYZER will send TABLE output to the queue assigned to the logical name ADM$SPOOL6, if SP=0 (the default) the ANALYZER queues output to ADM$SPOOL0. (See Section 21.1 "ADM$SPOOLn: Logical Print Queue Specification" for a generalized discussion of the ADM$SPOOLn logical name.)

12.19.11 Number of Copies

The Number of Copies option (NC) controls the number of copies that are printed when the ANALYZER is sending output to a print queue.

12.20 The STOP and QUIT commands

The STOP and QUIT commands are used to exit from the ANALYZER. Table output which has been redirected to a printer using the LP OPTION (see Section 12.19.7 "Line Printer") is queued for printing when either STOP or QUIT is used to terminate an analysis session.

12.20.1 STOP

The STOP command exits from the ANALYZER and insures that all of the new set building instructions, and option settings and definitional changes from the current analysis session are in the SAV file (see Section 12.2.2 "The SAV File").

12.20.2 QUIT

The QUIT command exits from the ANALYZER leaving the SAV file as it was before this use of the ANALYZER started. That is any new sets created, and option settings and definitional changes made in the current analysis session, are not written to the SAV file.

12.20.3 Exiting from the ANALYZER Via Ctrl/C

The user may abort the ANALYZER using Ctrl/C. Changes in the analysis session are written to the SAV file as they are performed. Hence even though Ctrl/C is used, the SAV file is "up to date". This is usually also the case if the user "bombs out", i.e. when an error exit from the ANALYZER occurs. However, when Ctrl/C is used to terminate the analysis, table output which has been redirected to a LIS file for printing is not queued.

12.21 Command Files

In addition to its interactive capabilities, the ANALYZER can receive its input from a command file. A command file can be named on the ANALYZER command line, or can be performed anytime within the analysis session. An ANALYZER command file may itself reference other ANALYZER command files.

If the command file is included on the ANALYZER command line, the default SAV file is initialized. Hence, command files invoked on the ANALYZER command line should start with the FILE command (see Section 12.2 "The FILE Command and SAV Files").

The "@" symbol indicates that a command file is being called. The following is the general syntax for calling a command file on the ANALYZER command line. If the file type is not included, the ANALYZER will look for ".COM".

```
$ an @command-file-name
```

12.21.1 Command File Conventions

Within a command file, the DISPLAY instruction at the beginning of a line will cause the text on that line to be displayed when the command file is being executed.

"DISPLAY n" displays "n" blank lines.

An asterisk (*) in the first column of a line in a command file will cause the text following it to be treated as a non-displayed comment.

The characters "CR" simulate a carriage return.

The PAUSE command may be placed anywhere within a command file, and causes the terminal to stop printing until the user presses return.

The BRIEF and VERIFY commands control which text from the command file is displayed on the terminal while a command file is being executed. In BRIEF mode no text, other than that generated by DISPLAY, is displayed while the command file is executed. VERIFY mode displays all the command text while the command file executes. The default mode for a command file is the VERIFY mode.

A command file can call another command file via the "@yyy" syntax, where "yyy" is the name of another ANALYZER command file.

12.21.2 Example of an ANALYZER Command File

The following ANALYZER command file, builds several sets, creates a field and prints a table.

```
* EXAMPLE.COM
*
DISPLAY   This command file build the sets to analyze
DISPLAY   maintenance history of our vehicles.
DISPLAY
FILE PREV.MAS
SELECT
PARTS GT 300 S:HIPARTS
LABOR GT 300 S:HILABOR
TOTMAINT GT 2000 S:HIMAINT
TOTMILES GT 50000 S:HIMILE
MODYR EQ 81 80 79 S:NEW
CR
*
COMPLEMENT NEW = OLD
CREATE AGE/I 82 - MODYR
TABLE
NEW OLD
HIPARTS HILABOR
#INT TOTMAINT AGE/AV
```

This ANALYZER command file is called as follows.

```
$ an @example
```

12.22 Multi-Value Data

The ANALYZER may be used on files that contain multiple values for a given category. An example of multi-value data is "language skills" in a personnel file or "courses taken" in a student record file. If "language skills" are coded in five fields named LS1, LS2, LS3, LS4, LS5, allowing up to five languages per person, and specific languages are given numeric codes, such that any of the language skills fields can contain any of the language codes, we could have the following personnel file definition data:

```
* PERS.DEF
MAS 1000
PERS# X9999 KEY1
...
LS1 I  "first language skill"
LS2 I  "second language skill"
LS3 I  "third language skill"
LS4 I  "fourth language skill"
LS5 I  "fifth language skill"
...
```

The language codes are as follows:

```
LANGUAGE CODES

1 English
2 French
3 German
4 Russian
5 Chinese
6 Danish
7 Swedish
8 Spanish
9 etc.
```

In order to build a set of employees who have language skills in French, we could do the following: First we create the virtual field "FR" (which does not exist in the personnel file) and set it equal to the code for French, which is 2.

```
AN> create fr/i 2
```

Then the SELECT command is used to find all the records in which the value of the field FR (2) is found in any of fields LS1, LS2, LS3, LS4 or LS5. This set is called FRENCH.

```
AN> select fr eq f:ls1 f:ls2 f:ls3 f:ls4 f:ls5 s:french
25  FRENCH (115)  11%
```

12.22.1 Building Sets from Multi-Value Fields

A second way of handling multi-value data is to place multiple codes into one alphanumeric field. This is the way the OUTPUT command (see Section 12.17.5 "Recoding Multi-Value Fields") recodes sets where the same item (record) can be in multiple sets.

In order to build sets from alphanumeric data coded with multiple values in each field, the character string operator IN (includes) can be used with the SELECT command (see Section 12.4.1 "Single Set Syntax"). For example, values for the language skills are the following:

Value	Language
EN	English
FR	French
GE	German
RU	Russian
CH	Chinese
DA	Danish
SW	Swedish
SP	Spanish
IT	Italian

Multiple values are stored within one field, e.g. LANG. The following data is from records coded with multi-value fields. The field LANG is an A30 field containing the codes for various language skills. PERS# is the employee number.

PERS#	LANG
0001	EN FR
0002	SP CH
0003	EN
0004	DA SW EN
0005	EN
0006	EN RU GE
0007	EN FR SP IT
0008	EN
0009	GE RU

For example, PERS# 0004 speaks Danish, Swedish and English.

We can build a set of employees who speak French (LANG code is FR), and a set of employees who speak English (LANG code is EN) as follows.

```
AN> select lang in FR s:french
AN> select lang in EN s:english
```

Note that the IN operator is case sensitive. Therefore, there has to be an exact match between the constant string (e.g. FR) and the data.

12.23 SPAWN

If the SPAWN command is called with a valid command line as an argument, spawn runs the command and then returns directly to the ANALYZER session:

```
AN> SPAWN DIR *.MAS
```

Chapter 13:Utilities

ADMINS includes several special purpose utility programs. They are:

PREPROCESS	Read an ADMINS instruction file as input, output the file with indirect references, conditional compilation logic, etc. resolved.
File Utility (AdmFU)	Provides brief or detailed information about ADMINS data files. Initialize an ADMINS data file.
AdmDIFF	Identify differences in the data contained in two ADMINS datafiles that have identical record structures.
FILECONVERT (AdmFcv)	Convert a keyed file to a sequential file.
SYNC	Provide synchronization between ADMINS commands.
AV (AdmAv)	Communicate with ADMINS data files via logical names.
PASSW (AdmPassw)	Provide password protection for an ADMINS file.
JOIN (AdmJoin)	Concatenates lines from a text editable input file and writes the resulting longer lines to an output file, which is also text editable.
AdmDate	Produces the current time and date in a standard format.

The following utilities allow the use of ADMINS logical names in places where the only alternative would be to use Windows commands that don't know anything about ADMINS logical names. Most of these utilities are also smart in that if they operate on an ADMINS data file, and that data file has text fields, they know how to deal with the .TCF and .TSF files that go with the data file.

AdmCpy	Copies files.
AdmDel	Deletes files.
AdmRen	Renames files.
AdmDir	Produces and handles logical names.

Each of these utilities is described in detail in the sections that follow.

13.1 PREPROCESS ADMINS Instruction File

PREPROCESS is used to produce an instruction file listing with all indirect references ("@@" files) included and all conditional compilation logic ("#ifdef" etc.) resolved.

- **Usage**: **preprocess** [options] [filename] [output filename]
- **Options**:

-@	Process using ANALYZER style include file syntax, i.e. lines that begin with one "@" are files to be included, rather than two ("@@").
-c	Used to specify a line continuation character.
-l	Print with line numbers. Line numbers for included files begin with l.
-p	Turn on parameterization. All substitutable parameters are processed (see Section 1.3.4 "Parameterization") (If standard output is redirected, this option is ignored.)

- **filename** - is an ADMINS Instruction file, or any standard ASCII text file. Files that do not make use of ADMINS' compiler syntax (e.g. indirect references, conditional compilation etc.) are simply displayed as-is, the same as if displayed by the TYPE command. PREPROCESS prompts for an ADMINS Instruction file if none is supplied
- **output filename** - File in which to write the results. If not supplied results are written to standard output.

13.2 AdmFu: ADMINS File Utility

The ADMINS File Utility (AdmFu) provides information about ADMINS data files in either brief or detailed format, and it can initialize an ADMINS data file, i.e. make it "empty".

13.2.1 AdmFu Dialogue

When called by itself on the command line, AFU prompts the user for the function to be performed:

```
$ AdmFu
16-Mar-93 14:31:07
Enter a Function; INIT, D, DD, Dxn, Dropn LP, TI, H:
```

The functions are:

INIT	Initialize an ADMINS file.
DUMMY	Initialize and add a dummy record to an ADMINS data file.
D	Summary description of an ADMINS Datafile.
DD	Detailed description of an ADMINS Datafile.
DXn	Disable Index n for an ADMINS Datafile (no n or n=A if all)
DROPn	Drop Index n for an ADMINS Datafile (no n or n=A if all)
LP	Direct Display Output to the ADM$SPOOL0 Printer
TI	Direct Display Output to the terminal.
H	“Help” information is displayed explaining the AFU functions.

Enter the code for the function you want.

If one or more valid file specifications are supplied on the AdmFu command line, AdmFu automatically performs a "describe" (function code "D") the files specified. When finished with the specified files, AdmFu prompts "Enter Datafile Name(s):" for more files to describe.

If the function code INIT, followed by one or more valid file specifications is supplied on the AdmFu command line, AdmFu assumes the specified files are to be initialized and prompts for confirmation (see Section 13.2.2 “Initialize”).

Each of these functions will now be described in detail.

13.2.2 Initialize

Use the "INIT" function to initialize an ADMINS data file, i.e. initialize the key index and the file control values in the header so that the file is emptied.

Example:

```
$ AdmFu
12-Feb-93 14:52:03
Enter a Function; INIT, D, DD, DXn, DROPn, LP, TI, H: init
Enter Datafile Name(s): re.mas
RE.MAS will be Initialized. Continue Initialize? y
Enter a Function; INIT, D, DD, LP, TI, H: cr
$
```

INIT can be invoked directly on the command line. For example, to initialize RE.MAS and RE.FLG:

```
$ AdmFu init re.mas re.flg
RE.MAS will be Initialized. Continue Initialize? y
RE.FLG will be Initialized. Continue Initialize? y
```

AFU prompts for confirmation before initializing each file.

13.2.3 Describe

The "D" function is used to describe an ADMINS data file. The description displays the following information:

1. date
2. time
3. structure level ("L:")
4. file name
5. number of stored records
6. number of available record positions left in the file[1]
7. record length in 16 bit words
8. number of fields per record
9. number of 1024 byte disk blocks reserved for the file
10. index pointer size (16-bit or 32-bit)
11. address of the root index block (relative to end of the file)
12. last index block used (relative to end of the file)
13. the SELECT expression from the DEF, if any

For a detailed explanation of these ADMINS data file characteristics, see Appendix E: “File Concepts”.

1. See Appendix E.4 “Available Space”

The "describe" function is called automatically by supplying a valid file specification on the command line. To get a description of DISPATCH.MAS[2]:

```
D:\TESTING\DATA>admfu dispatch.mas
Summary File Description 02-Mar-07 15:37:35    (File level 3)
File dispatch.mas
    ------------------------------------------------------
    Number of records stored                  272,506
    Number of records available               269,419
    Record size                                   390 words
    Number of fields                               76
    File size in 1024 byte blocks             424,819 blocks
    Index pointer size                             32 bits
    Index root pointer                        -24,347
    Last index pointer used                  -284,833
    ------------------------------------------------------
    Index 1: Calls_By_Status
             CLOSED AREA STATUS PRIOR RECDAT RECTIM
```

You can get descriptions of multiple files by entering multiple names on the command line (delimited by blanks).

At the end of the dialogue, use carriage returns to exit stepwise out of AdmFU.

13.2.4 Detailed Describe

"DD" is used to obtain a detailed description of the file. This description includes everything provided by function "D" (describe) plus the name, type, size, key/sort status, and picture for each field defined for the file. "DD" also shows the size of the DEF area of the file header.

```
C:\bills>admfu
02-Mar-07 16:32:57
Enter a Function; INIT, DUMMY, D, DD, DXn, DROPn LP, TI, H:  dd
Enter Data File Name(s):  po.mas
Detail File Description 02-Mar-07 16:33:00     (File level 2)
File po.mas
    ------------------------------------------------------
    Number of records stored                    53
    Number of records available                220
    Record size                                  8 words
    Number of fields                             4
    File size in 1024 byte blocks               11 blocks
    Index pointer size                          32 bits
    Index root pointer                          -1
    Last index pointer used                     -1
    ------------------------------------------------------

Press return to continue ...

   ----------------- ---- --------- ---- ----
   Field             Key/ Data      Byte Word
   Name              Sort Type      Size Size
   ----------------- ---- --------- ---- ----
   PO#               KEY1 Pictured  4    2    X99999
   DATO                   Long Date 4    2
   CNO                    Pictured  2    1    X9999
   AMT                    Decimal   6    3    2 Decimal Place(s)

Enter a Function; INIT, DUMMY, D, DD, DXn, DROPn LP, TI, H:
```

2.IThe file summary listing of AdmFU will list the file's Dictionary ID if the file has been defined from the ADMINS Data Dictionary.

13.2.5 DXn - Disable Index n

"DXn" is used to disable Index n for an ADMINS Datafile. Enter the following syntax to use this function:

```
no n
```

OR

```
n=A (if all)
```

13.2.6 DROPn - DropIndex n

"DROPn" is used to drop Index n for an ADMINS Datafile. Enter the following syntax to use this function:

```
no n
```

OR

```
n=A (if all)
```

13.2.7 Line Printer

"LP" is used to direct the information asked for to the "line printer",[3] rather than the terminal ("TI"), which is the default option. For an example, see Section 13.2.4 "Detailed Describe".

13.2.8 Terminal

"TI" is used to redirect the information asked for back to the terminal after "LP" has been used. Example:

```
$ AdmFu
02-Mar-07 15:36:55
Enter a Function; INIT, D, DD, DROPn, DXn, LP, TI, H: lp
Enter a Function; INIT, D, DD, DROPn, DXn, LP, TI, H: dd
Enter Datafile Name(s): re.flg
Enter Datafile Name(s): cr
Enter a Function; INIT, D, DD,DROPn, DXn, LP, TI, H: ti
Enter a Function; INIT, D, DD,DROPn, DXn, LP, TI, H: d
Enter Datafile Name(s): dispatch.mas

Summary File Description 02-Mar-07 15:37:35    (File level 3)
File dispatch.mas
    ------------------------------------------------------
    Number of records stored                  272,506
    Number of records available               269,419
    Record size                                   390 words
    Number of fields                               76
    File size in 1024 byte blocks             424,819 blocks
    Index pointer size                             32 bits
    Index root pointer                        -24,347
    Last index pointer used                  -284,833
    ------------------------------------------------------
    Index 1: Calls_By_Status
             CLOSED AREA STATUS PRIOR RECDAT RECTIM
```

3. "Line printer" is the queue pointed to by the logical name ADM$SPOOL0.

13.2.9 Help

"H" directs AFU to display the following "help" text describing AFU functions:

Available Functions:

INIT	Initialize an ADMINS Datafile
D	Summary Description of an ADMINS Datafile
DD	Detailed Description of an ADMINS Datafile
LP	Direct Display Output to the ADM$SPOOL0 Printer
DXn	Disable index n (for index number) or a (for all indexes) for an ADMINS Data file
DROPn	Drop an index n (for index number) or a (for all indexes) for an ADMINS Data file
TI	Direct Display Output to the Terminal

13.3 AdmDIFF: File Differences Utility

AdmDIFF identifies differences in the **data** between two ADMINS files of **identical structure.**

The two files being compared must have the same file structure, i.e. the same key structure and the same number of fields of the identical type in identical order. Field names are not checked and may differ. If the files contain TI or TX fields, AdmDIFF outputs a warning message that it does not check for differences in text fields, and continues processing.

AdmDIFF finds differences according to the key value in each file. If a record with a given key value exists in one file and not the other, AdmDIFF prints out the key value and the file name where it was found.

If a record with a given key value exists in both files, AdmDIFF compares each field in turn. If fields differ in value between the two files, AdmDIFF identifies the pair of fields and displays the two values.

When AdmDIFF reaches the end of both files, it outputs summary information including:

1. Total number of records found in file A and not in file B.
2. Total number of records found in file B and not in file A.
3. Total number of field differences found in 'n' records.
4. Total number of identical records.

The AdmDIFF command syntax is:

```
AdmDIFF[/TOTAL] FILE1 FILE2
```

The optional /TOTAL qualifier prints out only the summary information.

For best results, key values in both files should be unique.

You may compare files using an alternate index by using the index number for both files, e.g.:

```
ADMDIFF N.MAS-3 B.MAS-3
```

A sample AdmDIFF run follows:

```
$ AdmDIFF N.MAS XNA.MAS

KEY:      16 has field differences
Field FLD in A: ED
Field FLD in B:
Field D2 in A:                20.20
Field D2 in B:                20.00
KEY:      17 has field differences
Field FLD in A: DX
Field FLD in B: DF
KEY:      18 has field differences
Field FLD in A: SZ
Field FLD in B: XX
Field D2 in A:                 1.00
Field D2 in B:                  .00
KEY:      19 exists only in xna.mas


There are 0 records is n.mas that were not found in xna.mas.
There are 1 records is xna.mas that were not found in n.mas.
There are 3 records in both files that have 5 differences.
There are 109 identical records in both files.
```

13.4 FILECONVERT - Convert ADMINS datafile attributes

FILECONVERT (AdmFcv) is a multi-purpose file conversion utility that is used in several distinct ways to alter specific characteristics of an ADMINS data file.

13.4.1 Sequentialize an ADMINS data file

The FILECONVERT[4] command can be used to bypass the built-in index area of an ADMINS file if that area has become corrupted. Index corruption is experienced as ADMINS commands bombing out at a particular record in a file. In this situation, the FILECONVERT command can be used to alter the file header so that ADMINS thinks the file is a sequential file, i.e. a file that doesn't have a built-in index area. (See Appendix E for an explanation of the ADMINS file concepts underlying the use of FILECONVERT.) The FILECONVERT dialogue is as follows:

```
$ AdmFcv
File name:badfile.mas
BADFILE.MAS converted
File name:cr
$
```

After FILECONVERT has converted the corrupted file from a keyed file to a sequential file, the file should be readable. Then MOVE or SORT should be used to move the records out of the bad file into a new copy of the file.

Because of the nature of how ADMINS deletes and inserts records, described in Appendix E: "File Concepts", deleted and inserted records will appear differently after a file has been converted by FILECONVERT.

Deleted records reappear in the file. They must be re-deleted (or otherwise removed) using application level criteria, unless the file is a level 3 file (see Section 2.6 "Level 3 File Structure") where the deleted records do not come back.

Inserted records do not appear in sort order but in the chronological order of their insertion into the file. A SORT from the converted sequential file to a keyed file reestablishes the inserted records in their proper place.

4. FILECONVERT superseded the obsolete ADMINS/V32 SEQ command. It has identical syntax for the sequentialize function that was formerly performed by SEQ, and consequently, for compatibility with existing applications, is referenced by the symbol "SEQ" (as well as the symbol "FILECONVERT" in the ADMINS OpenVMS symbol definition command file, ADMSYMDEF.COM and in ADMINS Windows symbol definition file Adm_commands.fil).

13.4.2 Convert Structure Level

FILECONVERT is also used to "convert" ADMINS data files between structure levels. In most cases you need not be concerned about the structure level of an ADMINS file, however, for an explanation of those circumstances when structure level does become an issue, see Appendix E: "File Concepts".

The syntax for converting between structure levels is illustrated in the following example:

```
$ AdmFcv
File name:badfile.mas 1
Converting Level 2 file to Level 1
File name:cr
$
```

If the file name is followed by a space and a number, FILECONVERT converts the file to that structure level number (the possible values are 0, 1, or 2). FILECONVERT cannot convert between a level 3 and pre-level 3 files. The only solution is to define a version of the file of the desired level and MOVE or SORT the data into the file.[5]

Conversions between Level 0 and Level 1 modify an indicator in the file header and, if the key contains any numeric fields, FILECONVERT re-writes the file index (see Appendix E: "File Concepts"). Conversions between Level 1 and Level 2 only modify the file header, the index is not altered. FILECONVERT never alters the data in the file. **Conversions between Level 0 files and Level 2 files are not supported.**

If a Level 2 file uses any of the field types that are not available in Level 1 files (i.e. L, TM, DT), that file cannot be converted to a Level 1 file. If a Level 2 file is defined via the Data Dictionary and/or is defined with the /READONLY qualifier and the file is converted to Level 1, the Data Dictionary information and/or the read-only flag are retained and remain effective.

13.4.3 Convert Collating Sequence

To convert existing data files to 8-bit collating sequences (see Section 2.12 "Alternative Collating Sequences"), use the FILECONVERT utility's "C" option. On the command line or at the prompt,[6] enter the name of a file to convert followed by space and "C", e.g.:

```
$ AdmFcv dk.mas c
```

5. This new file system cannot be supported on older versions of ADMINS (available from V8.2 or higher). If you need to create files that can be accessed by pre-8.2 versions of ADMINS, use the **/ONEIX** command line switch when defining the file. A Level 2 file is created.
6. FILECONVERT can accept a list of files to process, which makes large scale data conversions easier (see Section 13.4.4 "Converting a List of Files")

The file is converted from its current collating order to the one indicated by the two character identifier stored in the logical name ADM$COLLATE.

```
$ AdmFcv tcol.mas c
Converting file from DEFAULT to DK collating.
File converted.
```

If ADM$COLLATE is not assigned, the file is converted to the default collating sequence, 7-bit ASCII (unless it already is 7-bit ASCII, in which case FILECONVERT just gives a message).

```
$ AdmFcv dk.mas c
File already collated using DEFAULT table. Nothing done.
```

If a file has as keys either alpha (An) fields, or Xpic fields which contain alphas (e.g., X99A99999), FILECONVERT automatically performs an index-only SORT after the data has been converted, to put the file into correct sort-order for the new collating sequence.

```
$ AdmFcv xcol.mas c
Converting file from DEFAULT to DK collating.
File converted.
XCOL.MAS will be sorted after FILECONVERT is finished
FILECONVERT finished.
Now executing FCVA3.COM...
SORT
Input file....: XCOL.MAS
Output file...: IX
Rebuilding index only. OK? Y
11:44:32.00
11:44:33.45 134 records read 8 blocks 1 section(s)
11:44:33.86 Index rebuilt
```

FILECONVERT does not make a copy of the file, it writes over existing data in the file. This saves disk space and time in large data conversions. **It is your responsibility** to make a backup copies of files which are to be converted. **You must do this,** because if something goes wrong (e.g., the system crashes midway through converting a file), the partially converted file is not useable and cannot be made useable. (The corrupted file has to be deleted, and the conversion must be restarted using the original file).

13.4.4 Converting a List of Files

FILECONVERT will accept a list of files to process. Prepare a list (using a text editor) that has a file name and a FILECONVERT option code on each line. The following shows the contents of a text file called "FCLIS.LIS":

```
JANTR.DER C
FEBTR.DER C
MARTR.DER C
APRTR.DER C
MAYTR.DER C
```

If we then use the syntax "@listfile" on the FILECONVERT command line, or at the FILECONVERT prompt:

```
$ AdmFcv @fclis.lis
```

FILECONVERT will perform each of the requested operations.[7]

13.5 SYNC - Synchronization Between ADMINS Commands

There is often a need, particularly in communication-based applications, to synchronize the activities of ADMINS commands. The SYNC command provides this capability to the ADMINS developer via the use of ten locks, numbered from 50 to 59.

7. If one of the requested operations is FILECONVERT option "C", FILECONVERT first does the collating sequence conversion (or any other FILECONVERT operation) on all files, and then sorts the files which require it (see Section 13.4.3 "Convert Collating Sequence").

The SYNC command is typically used in command files (this use is also described in Section 14.13 “Synchronization of ADMINS Command Files”). The same (simulated) event flags may be used to synchronize events in TRANS via the SYNC subroutine (described in Appendix H.14.17 “SYNC - Synchronize Access to a File”).

The syntax of SYNC is as follows:

```
$ sync [/system] ef action
```

"/SYSTEM is an optional parameter that, when used, indicates that the lock (or "flag") is to be in effect system-wide, rather than group-wide (the default).

"EF" is a "lock" or "flag" number between 50 and 59.

"ACTION" is either "W", "S", or "X" , specifying one of the following functions:

W: SYNC should test the status of flag EF. If EF is free then SYNC should take it, and allow the command file to continue running. If EF is already taken i.e. if SYNC running in another command file has flag EF, then SYNC should wait for flag EF to be released and then proceed as above to take EF and continue with the command file. (In this context "waiting" means suspending the SYNC command until the flag is released). For example:

```
$ sync 52 w
```

S: SYNC should free flag EF, i.e. signal availability. That is, this command would be placed following the commands in the command file that were being synchronized via a previous “SYNC EF W”. For example:

```
$ sync 52 s
```

X: SYNC operates as with the "W" code except if the flag is not available when first tested, then SYNC should make the command file exit, i.e. "abnormal termination". While the "W" action is used to wait for the other command file to release control of EF, the "X" action is used to cause the command file to terminate if another command file has control of EF. For example:

```
$ sync 52 x
```

13.5.1 FLAGS Utility

The FLAGS utility is used to inspect how the SYNC flags are being used.

```
$ FLAGS

   GROUP 0 Flags
  Flag       PID Host
    50      Free
    51      Free
    52  00000f54 soft.admins.com
    53      Free
    54      Free
    55      Free
    56      Free
    57      Free
    58      Free
    59      Free
```

13.6 AdmAv - Communicate with ADMINS Files via Logical Names

The AdmAv command provides a way for command procedures and shell scripts to directly access, read, and update ADMINS data files using logical names.

Before AdmAv is called you must assign logical names to specify what action is to be taken. Assign the file specification of the file to be accessed to the logical name A$FL (a_fl on Windows systems), and assign the code for the operation AdmAv is to perform to the logical name A$OP (a_op).

Code[8]	Operation requested. Valid operators are:
F or FL	Find the record by key; set the record position
R or RL	Read the record by record position
W or WL	Write the record by record position
C or CL	Clear logical name table of A$... or L$... entries
S or SL	Status information requested

AdmAv uses the logical name A$RP (a_rp) to hold the record position of the record accessed. A$RP is either set by AdmAv in a "Find record by key" operations or provided to AdmAv in a read/write operations that access the first (set A$RP to "FI") or last (set A$RP to "LA") record in the file. When A$RP is set to "FI" or "LA" for a read or write operation[9], A$RP resets to the record position of the first or last record in the file.

AdmAv uses logical names of the form A$fieldname (a_fieldname on Windows systems) to hold the values contained in the fields of the file being accessed. For example if fields in the file being accessed are named STREET, CITY, ZIP_CODE, and STATE, AdmAv uses the logical names A$STREET, A$CITY, A$ZIP_CODE, and A$STATE to hold the values of those fields.

8. Codes may be either upper or lower case.
9. A$RP settings are case insensitive.

If file status information was requested (i.e. if A$OP is set to "S") the following logical names are set by AdmAv:[10]

Logical Name OpenVMS (Windows)	Used to hold
A$NR (a_nr)	Number of records in the file
A$KL (a_kl)	Key length (in words)
A$RL (a_rl)	Record length (in words)
A$NB (a_nb)	Number of (1,024 byte) blocks in the file
A$LV (a_lv)	Index level
A$RA (a_ra)	Estimated number of records available in the file

```
N
```

Alternatively AdmAv utilizes logical names that begin with "L$" ("l_" on Windows) rather than "A$" to hold field and file status values. These "L$" logical names can then be used directly with parameterized instruction files (see Section 1.3.4 "Parameterization"). To instruct AdmAv to use L$ logical names rather than A$ logical names, append the character "L" to the operation code in the logical name A$OP.

For example, A$OP with the value "FL" will find a record by key (note: you would provide AdmAv with a key value using "L$keyfieldname" logicals) and load the values for its fields into L$fieldname logicals; A$OP with the value "SL" will create logical names LNR, LKL etc. AdmAv **always** uses the control logical names AOP, AFL, and A$RP.

Lets assume that we want to access the file CONTROL.MAS which has the following DEF:

```
MAS 100
*
ITEM   I   KEY1
OK     A1
COUNT  D
```

Assume that we need to get, in an ADMINS command file, the value of the COUNT field for a particular ITEM.

To accomplish this assign values to the appropriate logical names then call AdmAv ("#ifdef" conditional compilation syntax (see Section 1.3.6 "Conditional Compilation") is used to enable the ADMINS command file to run in either OpenVMS or Windows environments):

```
#ifdef $VMS$
$ ASSIGN  CONTROL.MAS  A$FL
$ ASSIGN  F            A$OP
$ ASSIGN  6            A$ITEM
#else
$ AdmLcr a_fl control.mas
$ AdmLcr a_op F
$ AdmLcr a_n  6
#endif
```

10. See Appendix E.4 "Available Space" for a discussion of "available space" in ADMINS data files.

```
AdmAv
```

Assigning the value "CONTROL.MAS" to the logical name A$FL (a_fl on Windows) identifies the file we want to access. Assigning "F" to A$OP (a_op) specifies the "find a record by key value" operation; and assigning 6 to A$N (a_n) specifies the key value we want to find (the record with a value of 6 for field N - the key field).

If an error occurs, for example, if the file specified in A$FL cannot be found, AdmAv exits with an error status set, and displays a diagnostic message. In addition, whenever a call to AdmAv results in an error condition it places a code indicative of the problem into the logical name A$AV_ERR or L$AV_ERR (a_av_err or l_av_err on Windows) The A$AV_ERR codes are:

```
 2 = No translation for mandatory logical name
 4 = File not found
 6 = Find with key value not set
 8 = Invalid record position (A$RP)
10 = Format error
12 = Invalid operation code (A$OP)
```

A normal result deassigns A$AV_ERR.

If the record specified in a "find by key" operation is not found the value "NF" is assigned to the logical name A$RP (a_rp on Windows). This is not an error (AdmAv exits with a normal status and A$AV_ERR is not set).

AdmAv assigns the value of each field in the record to the corresponding A$fieldname (a_fieldname) logical name. When AdmAv finds a field that has a null value (i.e. an Alpha field with a blank in it) AdmAv assigns the value "&&" to the corresponding logical name,[11] rather than a null string. Therefore to test a field for a null value, test for "&&" in the corresponding logical name, as follows:

```
$ IF "''F$TRNLNM("A$OK")'" .EQS. "&&"          (OpenVMS)
THEN...

$ if [ "`AdmLtr a_fld`" =  '&&' ] ...          (DCL Script
                                               Windows Perl
                                               Script)
```

13.7 AdmPassw - Password Protect An ADMINS File

A user may assign a password to a file. The password consists of eight or fewer characters. In order to assign a password to a file the AdmPassw command is used. The name of the file to be password protected may be included on the command line or AdmPassw will prompt for the name. AdmPassw then prompts for the new password for the file named. (Whenever AdmPassw prompts for a password the response typed by the user is **not** echoed on the terminal.)

```
$ AdmPassw emphist.mas
TYPE NEW PASSWORD: (not echoed to terminal)
FILE: cr

$ AdmPassw
FILE: emphist.mas
TYPE NEW PASSWORD: (not echoed)
FILE: cr
```

If the file already has a password, AdmPassw asks for the current password before accepting the new password. (A new password of nothing, i.e. carriage return, simply removes the current password from the file.)

```
$ AdmPassw emphist.mas
TYPE CURRENT PASSWORD:
TYPE NEW PASSWORD:
FILE: cr
```

Whenever a password is assigned to a file and ADMINS is asked to open the file, it requires that the user supply the password. The password can be provided in one of two ways.

1. The password can be appended to the full file name separated by a slash, as in the following examples where RE.MAS is the file name and REAL is the password.

11. If AdmAv is writing to a file (A$OP is "W") and "&&" is assigned to a logical name corresponding to one of the fields in the file, that field will be blanked out.

The FILE statement in a REP:

```
FILE RE.MAS/REAL

TRANS (General Editor Mode):

TRA RE.MAS/REAL
```

2. If ADMINS is not given the password appended to the file name then the ADMINS command will prompt for the password just before opening the file.

13.8 AdmJoin

The AdmJoin utility concatenates lines from a text editable input file and writes the resulting longer lines to an output file, which is also text editable.[12]

AdmJoin uses two delimiter characters specified by the user. Each line in the input file may end with one of these delimiter characters. The line segment delimiter immediately follows the last data character in an input line which is not the last line segment in an output line. The record delimiter immediately follows the last data character in an input line which is the last segment in an output line.

If there is one blank after a delimiter, the blank is ignored. If an input line does not end with either of the delimiters, AdmJoin acts as if it ends with a line segment delimiter.

Input lines should not be more than 255 characters long, including delimiters. However, there is no limit on the length of an output line.

Just typing "AdmJoin" at the command prompt gives a syntax help summary.

12. AdmJoin was formerly known as LJoin on pre-8.2 versions of ADMINS.

The syntax is:

```
AdmJoin -sD1 -rD2 input_file output_file
D1 is the segment delimiter character
D2 is the record delimiter character
```

Example:

```
AdmJoin -s@ -r$ j.lis x.lis
```

Input file j.lis contains the following:

```
this is                @
some stuff             @
and this is end of line $
start second line      @
middle of second line  @
here, now the end, bye! $
```

After running the JOIN command above, output file x.lis contains:

this is	some stuff	and this is end of line
start second line	middle of second line	here, now the end, bye!

13.9 AdmDate

The output from the AdmDate command is date and time on the following form:

```
Day Mon DD HH:MM:SS YYYY
```

e.g.

```
Fri Jul 09 15:12:331999
```

This format is the same across all operating systems, allowing your program to extract information from the string in a consistant manner.

13.10 AdmCpy

AdmCpy is an 'ADMINS savvy' file copy utility. The syntax is as follows:

```
ADMCPY   [/q][/y][/c][/s][/w] source[-w]   target[-w]
```

AdmCpy reports every file it copies unless the **/Q** option is present (**/Q** does not suppress other messages). If the target file exists, AdmCpy prompts you and asks if you wish to overwrite the existing file, unless the **/Y** option is present, in which case it silently overwrites the target files. Use **/C** to tell AdmCpy to continue copying a set of files (without error exit) if some error occurs for a particular file

The **SOURCE** may be the path of a single file, or a wildcard file spec containing a single *. It may contain an ADMINS logical name.

The **TARGET** can be the pathname of a single file or a directory. If the **SOURCE** is a wildcard, the **TARGET** must be a directory. It can contain an ADMINS logical name.

If the **SOURCE** file is an ADMINS file with text field(s), the corresponding .TCF and .TSF files is also copied.

Use **/S** (for Secure) to take out locks for ADMINS data files being copied to prevent the input file from containing partial updates, and the output file from being copied on top of an already open file. With the /S switch the input file will be locked single user, and the output file will have an Exclusive lock (the same as would happen with AdmSort. WIth **/S** the source file cannot already be open for write, and if the output file exists it cannot already be open at all. **/S** only works for ADMINS data files.

When using **/S** use **/W** to signal that if any of the files are open AdmCpy should wait and try again until the file becomes available.

If waiting to get a file lock for a file should not apply to both the source and the target file, use "-W" appended to the file name to specify waiting for that file.

13.11 AdmDel

AdmDel is an 'ADMINS saavy' delete utility. The general syntax is:

```
AdmDel                  file1  [file2 [...] ]
```

where each file spec may be a single file, or a wildcard. They may contain an ADMINS logical name. Each file spec must be separated by one or more spaces.

It should be noted that Adm2Perl, the ADMINS utility that translates ADMINS command files into Perl scripts will convert the OpenVMS syntax for the DEL command, which includes version numbers and are comma separated, into valid AdmDel syntax. That is, on Win32, the version numbers are removed, and the commas are turned into spaces before the command line is passed to AdmDel.

13.12 AdmRen

AdmRen is an 'ADMINS saavy' rename utility. The general syntax is:

```
AdmRen                  [/q]   oldname       newname
```

Both **OLDNAME** and **NEWNAME** may contain ADMINS logical names.

13.13 AdmDir

The AdmDir utility allows you to list directories using logical names.

Chapter 14:Command Files

The ADMINS command file facility, AdmCom, pre-processes and executes sequences of ADMINS, Windows, and Perl commands contained in a text file.

All ADMINS commands can be executed in command files, including TRANS.[1]

14.1 Preparing A Command File

Command lines and responses to any prompts that occur are placed into a file (usually via a text editor). The resulting file is usually named with the file type ".ACF", e.g. VENDOR.ACF.

The following ADMINS command file uses MOVE/SELECT to create a file of vendors whose address is in Massachusetts, and then runs the report STATEVENDOR.

```
comment on
     *          massv.com
     *
     * Use MOVE/SELECT to move records for
     * Massachusetts from VENDOR.MAS
     * to STATEVENDOR.MAS
     *
     *
     move/select
       VENDOR.MAS          ! "Input File" response
       MSTATE EQ 'MA'      ! "Select" response
       STATEVENDOR.MAS I   ! "Output File" response
       CR                  !"# to move" reponse
     *
     report statevendor
     *
```

The conventions for preparing a command file are as follows:

1. Command names start in column 1.
2. Responses to command prompts are indented on the lines following the command.
3. An asterisk in column 1 indicates the line is a comment.
4. The exclamation point "!" can optionally be used as a comment delimiter in ADMINS command files. The COMMENT ON keyword causes the exclamation point to be treated as a comment delimiter from that point on in the ADMINS command file. The COMMENT OFF keyword causes the exclamation point to be treated as a normal character again. Be default the exclamation point is treated as a normal character.

1. TRANS is still operated "manually", e.g. via the keyboard/mouse, when called in a command file. When TRANS exits, the command file continues.

5. CHECKSTATUS ON/OFF

   Normally, a script or procedure generated by COM checks the exit status of each command and exits with a message if the status indicates an error. However, not all errors should stop a command file.

   To provide control over the handling of command exit status, place the keywords CHECKSTATUS OFF and CHECKSTATUS ON around the steps where exit status checking should be disabled (CHECKSTATUS OFF never followed by CHECKSTATUS ON disables checking for the remainder of the command file).

   On Windows systems, by default, AdmCom places “status checking” lines in the output Perl script after each unindented line in the input ADMINS command file:

```
$cur_exit = $? >> 8;
if ($cur_exit == $fatal_exit) {
  system("AdmLcr $exit_sev $cur_exit");
   goto EOC4;
   }
```

   When CHECKSTATUS OFF is in effect AdmCom skips insertion of this "status checking" lines between steps.

6. The response text "CR" means carriage return and is interpreted the same as a carriage return in interactive mode. "CR" is used in the above example for the "# to move" prompt of the MOVE command.

7. If column 1 contains a "$" , then the command on the line is assumed to be a Perl command.

   Lines beginning with “$” are preprocessed by AdmCom like all other lines (e.g. “#defined” names are substituted, and “<>” parameters are processed.) But no RESTART label is generated (see Section 14.10).

   The “source” line (after substitutions from preprocessing as described above) is **not translated**. It is “passed through” to the output script or command file “as is”, so it must be a syntactically valid Perl command.

8. If column 1 contains a “.”, then that line is displayed immediately (without the period), and is not written in the comxx[.COM] file. This “dot” syntax provides an easy way to display explanations of command file prompts before the prompts occur (see Section 14.3.1), for example,

```
*
.Enter the code for your area of interest
.    Code  Subject
.    ----  -------
.     1    Geography
.     2    History
.     3    Entertainment
.     4    Sport
.     5    Language
.     6    Mathematics
TRA <Code>MENU
```

14.2 Executing A Command File

A command file is executed by using the AdmCom command with command file name as the operand[2]. For example:

```
> admcom statev
```

If the command file name is not included on the command line, COM prompts for the name as follows:

```
> admcom
 Command file: statev
```

AdmCom translates the ADMINS command file into Perl script[3] with a name in the form Axxxxx.pl. The translation includes reorganizing the commands for proper execution, removing all the comments, and interpreting the command file options described in Section 14.3 “Parameterization” through Section 14.12 “Terminating a Command File In MAINT and PROD”. Finally, the translated procedure is executed.

Appendix C.2 “Commands and Procedures” contains an example of a Perl script produced on Windows from an ADMINS command file.

14.2.1 Progress Bar in an ADMINS Command File

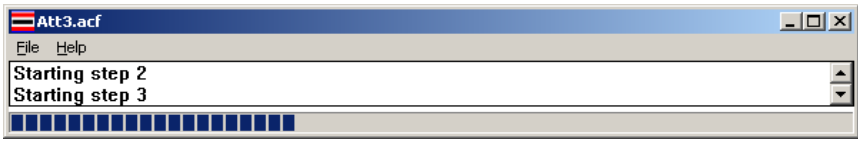
If AdmCom is run with the -P switch, for example:

```
ADMCOM  -P  ATT3
```

or

```
ADMCOM  -PX ATT3 (if you want the Progress Bar to close when the
Perl script finishes)
```

Adm2Perl generates[4] code to communicate the progress of the executing Perl script through AdmProgress.exe. AdmProgress.exe shows the progress of the command by displaying each step as it starts and also shows a scrollable listing of messages from the executing Perl script.



The communication between the executing Perl script and AdmProgress is done using a Named Pipe, a feature that is not supported under Windows 95/98/Me (i.e. it will only work on Windows NT/2000/XP).

PBARMSG statements (similar to DISPLAY statements) are now implemented in ADMINS command files to send specific messages instead of the generic "Starting step n", e.g.:

```
PBAR Sorting Att.Old into Att3.mas
```

---

2. If the command file name supplied does not have a suffix, it is assumed to be “.ACF”.
3. AdmCom produces a "Perl script". The "xx" part of the name of the file output is a unique automatically generated string - the resulting name is then assigned to the logical name adm_script (see Appendix C.1.1 “Differences in Print File and Temporary File Naming”).
4. AdmCom.exe calls Adm2Perl.exe, which parses the ADMINS command file and generates the Perl script. AdmCom then calls Perl to execute the Perl script. When Progress Bar is in use, the running Perl script is sending messages via the “named pipe” to AdmProgress.exe.

The PBARMSG statements are ignored if the command file is run without the -P switch.

The Progress Bar caption by default contains the name of the running command file (as shown above). You can specify a caption by appending =title to the -P (or -PX) command line switch, e.g.

```
AdmCom "-PX=Real Estate Billing" RECOM:REBill
```

Observe that the double quotes are necessary only if the title contains spaces.

14.2.2 Logging Output from an ADMINS Command File

If Admcom is run with the -log switch, for example:

```
ADMCOM -log ATT3   (output goes to ATT3.LOG)
```

or

```
ADMCOM -log=mydir:mylog.log ATT3 (to explicitly specify name
                                  for log file)
```

AdmCom sends the output that would ordinarily go to the Command Prompt Window to a text file.By default the command will have the same name as the ADMINS command file being run, with a file "type" or "extension" of ".LOG", e.g if "VENDOR.ACF" is run "VENDOR.LOG" would be created and receive the output from the command file.

To explicitly name the file to receive the command file output, append "=name" to the -log option, e.g. to run the "WEEKLY.ACF" ADMINS command file and save the output to the file "ASSESSOR_26.TXT in the folder identified by the logical name LOGDIR.

```
ADMCOM -LOG=LOGDIR:ASSESSOR_26.TXT  WEEKLY
```

14.3 Parameterization

COM allows parameterization of the text in the command file. Note that parameterization is evaluated **everywhere** in the command file. Therefore it may be used in both ADMINS and host operating system commands:[5].

5.If the logical name ADM_DIALOGBOX is set to the value “P”, command files (AdmCom) and reports (AdmReport) will prompt in a dialog box, rather than in the command prompt window.

The conventions for parameterization are the same as in REPORT (see Section 7.14 "Parameterization"). Parameters in single angle brackets ("< >") must receive a response or COM will exit without having created an output command file or script. Lines which include parameters with double angle brackets ("<< >>") that don't receive a response are ignored by COM in building its command file or script.

Once text has been supplied for a particular parameter, i.e. a particular angle bracketed string, then that text will be substituted for the parameter each time it is encountered, as is demonstrated in the following example.

14.3.1 Parameterization Example

The following command file is used to select records from any state from the vendor file for any year, and move them into the file STATEVENDOR.MAS.

```
*          statev.acf
*
* Use MOVE/SELECT to move records for
* state specified from the specified year's
* vendor file to STATEVENDOR.MAS
*
*
move/select
  <Enter year for vendor file>VENDOR.MAS
  MSTATE EQ '<Enter code for state>'
  STATEVENDOR.MAS I
  CR
  Y
report statevendor
*
<Enter code for state>
```

The complete dialogue of running STATEV.COM from terminal TTA3 would be as follows:

```
> admcom statev
Reading statev.acf
Enter year for vendor file: 93
Enter code for state: MA
A01971cd.pl written
A01971cd.pl started
Tue Jun 27 17:07:45 EDT 2006
----------------------------
move -select
Input file....: 93VENDOR.MAS
Select........: MSTATE EQ 'MA'
Output file...: STATEVENDOR.MAS I
# to move / S[kip] # / K[ey_range] / N[o_list]: cr
17:07:58
                                                                    *
100 records moved, total 100 records in STATEVENDOR.MAS
17:07:58
----------------------------
A01971cd.pl TERMINATED
Tue Jun 27 17:07:58 EDT 2006
```

14.3.2 Repetitive Parameterization

COM supports the repetitive parameters in REPORT[6] as described in Section 7.14.1 "Repetitive Parameterization". When COM finds a ~ (tilde) to the right of the double angle bracket (e.g. >>~), then COM reprompts and regenerates that line until the user replies with a carriage return to indicate that COM should proceed to the next line. COM will enter a line containing "CR" into the command file at the point where the

user replies with a carriage return. Since REPORT treats a "CR" in the command file as if the "CR" was a carriage return, the repetitive parameter sequence in the report instruction file is terminated and the next report instruction line is read.

In the following example a report includes a SELECT statement with repetitive parameters:

```
SELECT <<1 OR MORE SELECT EXPRESSIONS>>~
```

To run this report in a command file, the COM should include a repetitive parameter, i.e. a "tilde", e.g.:

```
REPORT PROMPTS
 <<TYPE DESIRED SELECT EXPRESSIONS, C.R. WHEN DONE>>~
```

The user enters the expressions to the prompts displayed by COM:

```
 > admcom prompts
<<TYPE DESIRED SELECT EXPRESSIONS, C.R. WHEN DONE>> acct eq 001
 <<TYPE DESIRED SELECT EXPRESSIONS, C.R. WHEN DONE>> month eq 2
 <<TYPE DESIRED SELECT EXPRESSIONS, C.R. WHEN DONE>> year eq 84
 <<TYPE DESIRED SELECT EXPRESSIONS, C.R. WHEN DONE>> cr
```

COM then builds the following lines into the command file to be run:

```
REPORT PROMPTS
 ACCT EQ 001
 MONTH EQ 2
 YEAR EQ 84
 CR
```

14.3.3 Logical Parameters

If a parameter string contained in the angle brackets begins with the characters “L_”, (e.g. <L_fieldname>), then AdmCom first tries to translate the prompt as a logical name. If the logical name has been assigned in either the process, desktop, or system logical name tables, the user is not prompted for the contents of the parameter. Instead the value of the logical name is substituted for the prompt. Parameters which begin with the characters “L_” and are assigned as logical names are called "logical parameters".

When the logical names exist, the display of logical parameter prompts and their values can be suppressed by assigning the lowercase letter "c" to the logical name OPTION (see Appendix A: “Options”).

If a parameter beginning with “L_” is not assigned as a logical name, then the user is prompted for a value as in standard parameterization (see Section 14.3 “Parameterization”).

Prompting for values when the logical name is not assigned can be avoided entirely by supplying a default value in the parameter string, as follows:

```
<L_MINIMUM=0>
```

Specify the default value for the logical name by appending “=value” to the logical name inside the angle brackets. In the example above if the logical name L_MINIMUM is not assigned, the value “0” will be substituted for the parameter.

For example, if the logical names "L_INFILE" and "L_OUTFILE" are assigned as follows:

---

6. Repetitive parameters in command files should only be used in conjunction with the repetitive parameter facility of REPORT. It is not implemented as a facility for general use in command files.

```
> admlcr 'l_infile'  "po.mas"
> admlcr 'l_outfile' "vendor.mas"
```

then a command file with the following statements:

```
admsort
  <L_INFILE>
  <L_OUTFILE>
```

will cause the PO.MAS file to be sorted into VENDOR.MAS.

If a logical name beginning with "L_" is used inside repetitive parameters (in conjunction with REPORT, see Section 14.3.2), then COM tries to translate a series of logical names. For example, if <<L_SELECT>>~ appears in a COM instruction file, COM tries to translate the logical name L_SELECT1, L_SELECT2, etc., until L_SELECTn is not found. These values are substituted in the command file. When L_SELECTn is not found COM will pass a "CR" to REPORT to signal the end of responses for that parameter.

To illustrate the use of this facility, we use the example from Section 14.3.2, which discussed repetitive parameters.

The repeating SELECT statement in the report remains the same:

```
SELECT <<1 OR MORE SELECT EXPRESSIONS>>~
```

But this time the command file used to run the report will utilize logical parameters:

```
REPORT PROMPTS
  <<L_SELECT>>~
```

If the following logical name assignments are made:

```
> admlcr l_select1 "acct eq 1"
> admlcr l_select2 "month eq 2"
> admlcr l_select3 "year eq 98"
```

COM then builds a command file identical to the one built in the example from Section 14.3.2:

```
REPORT PROMPTS
 acct eq 001
 month eq 2
 year eq 98
 cr
```

Note that COM inserts a "CR" to be passed to REPORT when L$SELECT4 is not found. If the logical name L$SELECT1 does not exist, then COM prompts the user for L$SELECT1, L$SELECT2, etc. until the user presses RETURN to the prompt (i.e. COM reverts to standard repetitive parameterization).

14.3.4 VALIDATE statements in ADMINS Command Files

Validate statements provide the capability to check the validity of responses given to parameters before they are actually used in processing.

If the VALIDATE statement expression is true, COM continues processing. If COM is running in interactive mode, and the VALIDATE expression is false, COM displays a user specified error message and reprompts.[7] VALIDATE will then test the next response.

VALIDATE statements automatically validate the data type (numeric, date, etc.) and length (alphanumeric) of the response and support the following comparison operations:

GT	Greater than.
GE	Greater than or equal to.
EQ	Equal to.
LE	Less than or equal to.
LT	Less than.
NE	Not equal.
BET...AND	Between (value) AND (value).

VALIDATE statements also support an "includes" operator, INCL, and a FILE operator.

The use of INCL is illustrated by the following example:

```
VALIDATE <Enter input file name>/A20 INCL "FY88"
&& Input file is not for Fiscal Year 1988!
```

which checks that the response to "Enter input file name:" includes the string "FY88".

The FILE operator indicates that the parameterized responses are to be combined to form the key to be searched for in the file specified, i.e.

```
VALIDATE <L$USER>/A10 FILE AUTHORIZE.TAB
&& Authorization Denied
```

checks that the value substituted for the parameter L$USER identifies a record in AUTHORIZE.TAB (and if that value is not in the file the command file will not be run).

The Boolean NOT operator can be used with any of the other supported operators:

```
VALIDATE <Account Number>/XA99999 NOT FILE CUSTOMER.MAS
&& Account already exists.
```

That is, if the Account Number entered is found in the file CUSTOMER.MAS, do not continue to process the command file.

7. Because COM reads through the ADMINS command file in a sequential manner and substitutes responses for prompts as they are received, VALIDATE statements should always precede the prompt they are checking. Otherwise the VALIDATE statement would re-prompt if its expression evaluates to false, but the prompt it is checking would already be substituted and incorporated into the output script.

14.3.4.1 VALIDATE Statement Syntax

VALIDATE statements have the following generalized form:

```
VAlidate parameter/type [ parameter/type][expression] &&message text
```

Each of the components are now described:

parameter	The ADMINS parameter being checked for validity.
/type	Any valid ADMINS data type specification. Data type must be supplied with a parameter. Data type must be separated from the parameter by a slash character, '/'. Use quotes (single or double) to surround parameter/type when the response may have imbedded blanks, i.e.: `VALIDATE "<Enter Name>/A40" EQ "<L$NAME>"`
expression	One of these comparison or misc. operations:
BET...AND	Between...and (<Enter Item No:>/X999 BET 100 AND 499)
GT	Greater Than (<Enter Date>/DA GT 1-JAN-88)
GE	Greater Than or Equal To
EQ	Equal To. operand
LE	Less Than or Equal To
LT	Less Than
NE	Not Equal To
INCL	Includes 'text'. Searches for 'text' within the parameter. (<Enter input file name>/A20 INCL "FY88")
FILE	Use Parameter(s) as key(s) to a record in the specified file. The datafile is opened read only. (<Account No. >/X999 NOT FILE CUSTOMER.MAS)
&&message text	Message text is displayed when statement evaluates false. '&&' is required, indicating error message existence to COM. '&&' is not shown in actual message display.

Parameters are always to the left of the operator. Multiple parameter values within a single VALIDATE statement are used only with the FILE operator (to build the entire key identifying a record) or when doing simple data type checking (i.e. no comparison operator is used).

14.3.4.2 VALIDATE Statement Examples

VALIDATE statements are designed to take advantage of the fact that each unique parameterized string in the COM file is prompted for only once, and the response to that prompt is substituted for that parameter everywhere it appears in the COM file. This is illustrated in the following example:

```
* post.com
*
* Command File to Post Weekly Transactions
*
VALIDATE <Enter Week Number>/I BET 1 AND 53
&& Week number must be in range 1 to 53
*
PROD
 <Enter Week Number>wk.mas
 #ACCOUNT
 PAYAMT PAYDATE
 post.rmo WI
 #ACCOUNT
 PAID PAYDT
 CR
*
DISPLAY POSTING OF TRANSACTIONS
DISPLAY FOR WEEK <Enter Week Number> COMPLETED
*
* end of post.com
```

The following are examples of some other VALIDATE statements uses:

```
VA <Enter 1st Parameter Value>/D2 <Enter 2nd Parameter Value>/A10
     &&INVALID PARAMETER DATA TYPES
```

The VALIDATE statement above simply validates that the responses are the correct data type.

```
va <Enter Printer Number>/I bet 0 and 9
&&Invalid Printer Device
```

Note that "va" (not case sensitive) is sufficient to identify a VALIDATE statement.

```
VALIDATE "<Enter Full Name>/A40" NE " "
&&Must Supply Customer Name
```

Note that quotes (single or double) must surround parameter/type if the response can contain imbedded blanks. The above parameter could produce the following COM dialogue:

```
 Enter Full Name: CHARLES C. COSMOS
```

the substituted form of the VALIDATE statement would become:

```
VALIDATE "CHARLES C. COSMOS/A40" NE " "
```

without the quotes COM would exit with an error message because CHARLES is not followed by a "/type".

14.3.5 CAPS ON/OFF: Convert Param Response to Uppercase

"CAPS ON" instructs COM to convert all[8] substitutions for parameterized prompts to upper case. "CAPS OFF" disables the effect of "CAPS ON", i.e. any parameterized prompts encountered after "CAPS OFF" will not be converted to upper case.

In the following example CAPS ON is used to ensure that the response to the prompt "<Select Town>" will always be upper case, then CAPS OFF is used to allow uppercase/lowercase responses for any subsequent prompts.

```
.
.
CAPS ON
MOVE/SELECT
  CONTACTS.MAS
  HOMETOWN EQ '<Select City or Town>'
  SELTOWN.DER
  CR
CAPS OFF
.
.
```

14.3.6 Disabling Parameter Parsing: Treat <string> As Normal Characters

If the backslash character (\) is present in the string assigned to the logical name OPTION, then AdmCom will not treat a string enclosed in angle brackets as a substitutable parameter if it is immediately **preceded by a backslash character**. Consider the following example:

```
display \<this is not a substitutable parameter>

display <this is>
```

If "\" is in OPTION, the first display statement will not prompt for a response and will display as-is (with the backslash removed). The second display statement will prompt for a response as usual.

This capability is especially useful when including Perl "file handle" syntax (which are also strings enclosed by angle-brackets) in ADMINS command files, for example:

```
$ open (FILE,"explanation.txt");
$ while (\<FILE>) {print}
admmove
   n.mas
   xxx.mas I
   CR
display done
```

8. All substitutable parameters, including those satisfied via "L_" logical names, are converted to upper case.

14.4 DISPLAY and PAUSE Statements

The DISPLAY and PAUSE statements are used to display messages on the user's screen while executing the command file, and to allow the user to terminate the run of a command file. DISPLAY displays a message and continues, whereas PAUSE displays a message, prompts for a response ("ANS:"), and terminates the command file unless the user responds with a "Y" to the prompt.

```
DISPLAY message
PAUSE message
```

The following command file selects records with errors from a time card file. If any records are selected, the user may respond with a "Y" to the "ANS:" prompt to run the report.

```
comments on
*      ERRORS.ACF
*
move
 time.mas                   ! time card file
 err.mas                    ! select bad records
 CR                         ! answers "# to move" prompt
display if there are any bad records
pause type 'y' to print them
report errors
```

The actual execution of the command file is as follows:

```
> admcom errors
READING errors.acf
A019a163.pl written
A019a163.pl started Tue Apr 25 12:19:25 2006
          ---------------------------
move
Input file....: time.mas
Output file...: err.mas
# to move / S[kip] # / K[ey_range] / N[o_list]: CR
12:19:31
************************************************************
0 records moved, total 0 records in err.mas 12:19:34
if there are any bad records
type 'y' to print them [Y/N]:
```

At this point the command file waits until the user has responded to the "ANS:" prompt.

```
type 'y' to print them [Y/N]: n
---------------------------
A019a163.pl TERMINATED
Tue Apr 25 12:19:42 2006
```

14.5 Translate Logical Name ADM$TERM: TT

When COM encounters the string "TT" (or "tt") anywhere in an ADMINS command file, COM substitutes the string assigned to the logical name ADM$TERM[9] in its place. "$TT$" provides an easy way for command files to use names for files that are unique and can be determined at run time.

The following ADMINS command file uses MARK.RMO to mark records in a copy of ACCT.MAS, and then sorts the file into a copy of MARK.IDX, on which a report called MARKED ACCOUNTS is run.

```
COMMENTS ON
     *
     * Make a copy of ACCT.MAS
     *
     COPY ACCT.MAS ACCT$TT$.DER
     *
     * Compile MARK to run on the copy
     *
     CMP MARK
      ACCT$TT$.DER                ! file-name parameter in the RMS
     *
     * Run MARK on the copy
     *
     MAINT MARK
      N                           ! no, not test mode
      Y                           ! yes, continue
     *
     * Set up a file to sort ACCT$TT$ into
     *
     COPY MARK.IDX MARK$TT$.IDX
     *
     SORT
        ACCT$TT$.DER              ! input
        MARK$TT$.IDX I            ! output (initialize)
     *
     REPORT MARKED ACCOUNTS
      MARK$TT$.IDX                ! file-name parameter in the REP
```

9. On Windows systems the ADM$TERM logical name is not used internally, but it is available for use in applications, as here with "TT". (It must be assigned explicitly - ADMINS does not assign it.)

14.6 xxx: Special variables

AdmCOM for Windows supports a number of special variables or tokens besides TT. Strings that appear anywhere in an ADMINS command file that begin and end with a dollar sign ($) are evaluated and have specific values substituted for them when the command file runs. These special variables are:

$TODAY$	will be replaced with today's date in the form YYYYMMDD (e.g. 20040422).
NOW	will be replaced with the current time in the form HHMMSS (e.g. 131005).
$TIME$	will be replaced with the current date and time in the form YYYYMMDDHHMMSS (e.g. 20040422131005).
$*string*$	where ***string*** is any **environment variable**, and will be replaced by that environment variable's value (e.g. if your login name is Peter $USERNAME$ will be replaced by "Peter"). If ***string*** does not exist as an environment variable, $***string***$ is left unchanged.

14.7 SKIP Part Of A Command File

The SKIP statement in a command file causes COM to skip all the lines in the ADMINS command file between the SKIP statement and the END statement. Typically, this is used with parameterization to skip over parts of a command file which may not be needed in a particular run.

For example, the following record maintenance procedure is generalized to set a field in a file with a value.

```
*         SET.RMS
FILE <FILE-NAME>
PROGRAM
<FIELD-NAME> = <VALUE>
```

If this RMO is used to set a key field value the file will need to be sorted. If a non-key field value is set no sort is necessary. A SKIP statement (entered via parameterization)[10] is used to handle the two alternatives:

```
*         SET.COM
CMP SET
* The following three parameters are passed to SET.RMS
 <FILE-NAME>
 <FIELD-NAME>
 <VALUE>
*
MAINT SET
 N                        ! no, not test mode
 Y                        ! yes, continue
*
```

10. Two levels of parameterization are used in the example: in the command file and also in the record maintenance procedure.

```
<<TYPE 'SKIP' IF KEYS WERE NOT CHANGED>>
*
SORT
 <FILE-NAME>
 CR                          ! no output file, i.e. a self sort
 Y                           ! yes, continue
*
END                          ! scope of the SKIP statement
```

14.8 Indirect Command Files

Command files can indirectly reference other command files. "@@" in columns 1 and 2 signifies a reference to another command file. There may be up to five levels of indirect references.

All indirect reference substitutions are made by COM **before** preprocessing.

For example, T1.COM contains an indirect reference to T2.COM, which in turn contains an indirect reference to T3.COM:

```
*  t1.com       *  t2.com       *  t3.com
*               *               *
afu m.mas       afu n.mas       afu o.mas
@@t2.com        @@t3.com
afu q.mas       afu p.mas
```

When T1.COM executes, the result is equivalent to executing a single command file that contained the following commands:

```
afu m.mas
afu n.mas
afu o.mas
afu p.mas
afu q.mas
```

14.9 BRIEF and VERIFY

The BRIEF and VERIFY statements control the amount of messages displayed during command file execution. VERIFY mode is the default mode and displays the complete output of the ADMINS commands. In BRIEF mode, only the command name is displayed. The BRIEF and VERIFY statements are placed in the text of the command file to turn "brief mode" on and off at that point in the processing of the command file. The ADMINS command file pre-processors, NATCOM (OpenVMS) and Adm2Perl (Windows), translate the BRIEF and VERIFY statements into the appropriate assignment of the logical name BRIEF: "Y" to turn BRIEF mode on, "N" to turn it off.

ADMINS commands check[11] the logical name BRIEF to determine whether to echo command lines and display prompts and messages. If "Y" is assigned to the logical name BRIEF then ADMINS operates in brief mode, i.e. ADMINS commands suppress output of their normal prompts and messages. In command files the ADMINS command line is echoed whenever one is called, even with "Y' assigned to

BRIEF. To suppress both command line echoing in command files as well as ADMINS command prompts and messages, "0" should be explicitly assigned to the logical name BRIEF:

```
> admlcr brief 0
```

If BRIEF does not translate to "Y" or "0" then ADMINS operates in verify mode.

14.10 Command File Calling A Command File

A command file can call another command file.

When a command file is used to call a command file, the last executed command in a command file may be another COM command to translate and execute another command file. The command file name of the second command file is required on the command line (e.g. "COM DOIT"). However, the use of the COM command in a command file does not stop the translation of the command file. Therefore, any number of COM commands may be included in a command file with the necessary logic to execute a specific COM command as the last executed command in the command file.

An important use for command files calling command files is to allow the operator to supply parameters to a second command file after viewing the results of the first command file.

14.10.1 Command File Calling A Command File Example

An interesting application of this feature is to use REPORT in a command file to create and then execute another command file. (When using REPORT to build a command file use "LENGTH 0" to suppress all carriage control characters.) For example:

```
*              RUNCOM.COM
*
REPORT MAKCOM     ! run the report, which outputs a
*                   file called DOIT.COM
*
COM DOIT          ! execute the DOIT command file just created
```

14.10.2 Command File Menu Example

Another use of including a COM command in a command file is to set up a simple menu command file for all the processes of an application. Then, by calling the menu, the user could execute any applicable process. For example:

```
*    MENU.COM
*
*  Display the list of processing choices available
*
```

11. Brief mode is intended primarily for use in command procedures, but the setting or the logical name BRIEF is checked and will have effect outside command procedures as well.

```
CLR
DISPLAY          Accounts Receivable Processing Menu
DISPLAY          ----------------------------------
DISPLAY
DISPLAY           Enter "D" for Daily Processing
DISPLAY
DISPLAY           Enter "W" for Weekly Processing
DISPLAY
DISPLAY           Enter "M" for Monthly Processing
*
COM CHOOSE ; call another command file to make the choice
```

Because the choice is parameterized in the next command file, the operator requests the running of PROCD.COM or PROCW.COM or PROCM.COM. The actual name of the Daily Processing command file is PROCD.COM, etc.

```
*    CHOOSE.COM
*
*  Select via parameterization the process to run
*
COM PROC<Enter Your Choice> ! execute chosen process
```

14.11 Restarting Command Files

ADMINS command files that abnormally terminate can be restarted after the cause of the failure exit is repaired. This capability is enabled by placing the keyword "RESTART" as the **first line** in the command file. Then if an abnormal termination occurs, the generated perl script (e.g. Axx.pl) is **not** deleted. After repairing the problem the operator can continue execution of the command file at the point it was terminated by re-rerunning comxx[.COM] as follows:

```
> perl a002c0e9.pl
```

When the RESTART statement is the first line of the command file, a RESTART point is set up **before each ADMINS command** in the generated script. An ADMINS command is defined as any unindented line that does not begin with a dollar sign ($).

14.12 Terminating a Command File In MAINT and PROD

A running command file may be terminated by a running MAINT or PROD. This is done by having the RMO set the E$XIT/I local field to "1". The command file will then terminate at the end of the step containing the MAINT or PROD. Section 10.7 “Terminating a Command File: E$XIT” describes the use of E$XIT in MAINT and Section 11.14 “Terminating a Command File: E$XIT” describes the use of E$XIT in PROD.

14.13 Synchronization of ADMINS Command Files

Often there is a need to synchronize the actions set out in ADMINS command files with other activities that are going on simultaneously in your organization. For example, you may want to ensure that a command file that produces a report does not run until posting of results to all appropriate files is complete. ADMINS uses a set of ten special "locks" (or "flags") numbered 50 through 59 for this purpose.

The SYNC command[12] is typically used in ADMINS command files to manipulate these locks to synchronize events, as follows:

```
...
*
* Wait for the availability of flag 52 before proceeding
*
SYNC 52 W
*
* Access the synchronized files
*
...
*
* Reset flag 52 so it is available to other processes
*
SYNC 52 S
...
```

14.14 Command File Communication: AdmAV Command

The AdmAV command allows for communication, using logical names, between a running command file and ADMINS data files. AdmAV can find records by key value and read or write the record found. AdmAV can also retrieve file status information about an ADMINS data file such as file size in blocks, number of records stored, etc.

See Section 13.6 “AdmAv - Communicate with ADMINS Files via Logical Names” for details on AV syntax and use.

14.15 Localized Command File Extensions

Three special logical names allow developers to introduce customized code into an ADMINS command file at startup and termination. These are:

12. The SYNC command is described in Section 13.5 “SYNC - Synchronization Between ADMINS Commands”.

- ADM$COM_STARTUP - identifies a PERL script that executes just prior to executing the actual content of the ADMINS command file, and may be used to:
  - log startup information like the value of the logical names, start time, user name, etc.
- ADM$COM_NORMALEXIT - identifies a PERL script that executes if the PERL script exits normally (at the EOC: label).
- ADM$COM_ABNORMALEXIT - identifies a PERL script that executes if the PERL script exits abnormally (at the EOC4: label).

The last two may write relevant information into the log file, mail status information to the user and/or support personnel, etc.

If these logical names exist when Adm2Perl is run, it creates a script with the following general structure:

```
use Cwd;
use File: : Basename
$ENV{‘ADM_INCOM’}=”Y”;
$exit_sev=”ADM_EXIT_SEVERITY’;
$fatal_exit=1;
system (“AdmLcr $exit_sev 0”)
($adm_cwd = wed()) =~ tr/\//\\/;
$last = substr($adm_cwd, -1, 1);
if ($last eq ‘\\’) { chop $adm_cwd; }
system(“AdmLcr ADM_SCRPATH $adm_cwd\\Affffffff.pl”);
$| = 1;
print(“ Affffffff.pl started “);
$dttm = localtime(time()); print $dttm, “\n”;
    <= Code from ADM$COM_STARTUP goes here
print(“----------------------------\n”);
    <= The content of the ADMINS command file goes here.
EOC:
print(“----------------------------\n”);
print(“Affffffff.pl TERMINATED\n”);
$dttm = localtime(time());print $dttm, “\n”;
system(“admldl RESTART_ffffffff’);
    <= Code from ADM$COM_NORMALEXIT goes here
system(“AdmDel ADM_SCRPATH”);
exit 0;
EOC4:
print(“----------------------------\n”);
print(“ABNORMAL TERMINATION Affffffff.pl\n”)
    <= Code from ADM$COM_ABNORMALEXIT goes here
$dttm = lcoaltime(time()); print $dttm, “\n”;
$cur_exit = ‘AdmLtr $exit_sev’;
exit $cur_exit;
```

14.16 PERL_OPTION Logical Name: Pause Before Exit at Abnormal Termination

When an ADMINS command file is spawned from another process (e.g. TRANS), it will run in its own Command Window. If the command has a problem and exits with “Abnormal Termination” the command file’s window ordinarily would disappear immediately, preventing the user from seeing any diagnostic messages. If the value

"W" is assigned to the logical name PERL_OPTION ADMINS command file's will wait for the user to type a carriage return before exiting after an Abnormal Termination, providing an opportunity for inspection of diagnostic messages.

Chapter 15:Basic RMO Functions with TRANS

A record maintenance procedure (RMO)[1] may function "behind", (i.e. along with,) an operating screen (a TRO). Some of the functions which can be performed by the RMO behind the screen include: calculations, changing values, data entry validation, and simulating the user entering keystrokes. Chapter 15: "Basic RMO Functions with TRANS" describes how TRANS communicates with the RMO and includes several simpler examples of RMOs used together with screens. Chapter 16: "Advanced RMO Functions with TRANS" describes the more complex uses for the RMO behind the screen.

The RMO is compiled separately from the screen description (TRS) with which it operates. The RMO-NAME is included on the header line of the TRS (see Section 5.3 "Screen Header Line") to identify the record maintenance procedure which operates with the screen. The RMO is written to operate on the same master file as the screen, and typically performs some operation on the active record and its link fields depending on what the user has just done at the terminal keyboard. For example, the following screen header line shows that PERSNL.RMO is to operate behind the screen PERSNL.

```
PERSNL PRF.MAS 1 PERSNL.RMO INSERT DELETE NOMSG
```

PERSNL.RMO **must** reference the same master file, using exactly the same file specification used in the screen header line.

```
FILE PRF.MAS
LOCAL
...
```

When the file specification in the TRS includes a device name, directory name and/or logical name, the file specification in the RMS must also include the device name, directory name and/or logical name.[2] For example a TRS with the following header line:

```
MODBUD DUA1:[FINANCE]BUDGET.MAS 1 MODBUD.RMO INSERT NOMSG
```

has an RMO with the following file specification:

```
FILE DUA1:[FINANCE]BUDGET.MAS
LOCAL
...
```

1. A record maintenance instruction file has the file type RMS. The compiled version of an RMS, prepared by the CMP command (described in Chapter 9: "CMP: The Record Maintenance Compiler"), has the file type RMO. The general ADMINS term for a record maintenance procedure is "RMO" although "RMS" is sometimes used as well.
2. File access options appended to the file name in the RMS are ignored in TRANS. File access options for the screen's main file are specified in the screen header line (see Section 5.3 "Screen Header Line").

15.1 Communication with TRANS

The record maintenance procedure used behind the screen should contain two local fields, S$S and M$M, which stand for "status" and "mode" respectively. These two fields are used by TRANS to "communicate" with the RMO. Each time TRANS executes the RMO (referred to as an "RMO call"), TRANS sets "status" and "mode" so that the RMO knows exactly what TRANS is doing. The values of M$M and S$S may **not** be changed in the RMO. That is, the RMO may not set M$M or S$S to another value. Only TRANS can set the values of these fields. The logic in the RMO can check for specific values in M$M and/or S$S (see Section 15.4 "Examples Using an RMO Behind the Screen") to determine which statements to execute in a particular RMO call. S$S usually has an "A6" field type[3] and M$M has an "A2" field type. They are included in the RMO as follows:

```
...
LOCAL
S$S/A6
M$M/A2
...
```

15.1.1 Status: S$S

The "status" of the call to the RMO by TRANS indicates that TRANS is in one of three conditions:

1. The RMO is being called before a record is to be displayed on the screen. The field S$S contains the string "BEGREC" ("beginning of record").
2. The RMO is being called after a value has been entered into a field (but before it is accepted and displayed). The field S$S contains the first six characters of the field name into which data has just been entered on the terminal.
3. The RMO is being called before a record is to be cleared from the screen. The field S$S contains the string "EOFREC" ("end of record").

When the RMO is called, the user may test S$S for "BEGREC", "fieldname", or "EOFREC" as follows:

```
...
IF S$S EQ 'BEGREC' THEN ...
IF S$S EQ 'SS#' THEN ...
IF S$S EQ 'EOFREC' THEN ...
...
```

3. S$S may be specified with a field type larger than A6 to accommodate field names that are not unique in their first six characters. As the largest field name permitted is 18 characters there is no need to make S$S larger than A18.

15.1.2 Mode: M$M

The "mode" of the call to the RMO refers to the current operating mode of TRANS as described in Section 6.3 “TRANS Modes”. M$M is set to a code for each mode as follows:

```
"UP" if TRANS is in Update Mode
"AP" if TRANS is in Append Mode
"IN" if TRANS is in Insert Mode
"ER" if TRANS is in Error Mode
"DE" if the active record is being deleted
```

When the RMO is called, the user may test M$M for any of the codes as follows:

```
...
IF M$M EQ 'UP' THEN ...
IF M$M EQ 'AP' THEN ...
...
```

Of course, the mode and status may be tested together.

```
IF S$S EQ 'BEGREC' AND M$M EQ 'UP' THEN ...
```

Reviewing then, when the RMO is called S$S contains "BEGREC", "fieldname", or "EOFREC". M$M contains either "UP", "IN", "AP", "DE" or "ER" for Update, Insert, Append, Delete or Error Mode.

The modes just described are set when the RMO is called after all LINK paragraphs in the screen description have been executed. This call is referred to as the **"post-link"** call and follows execution of the links so that the RMO may use the field values fetched by the links. The post-link calls are the RMO calls most commonly used by the screen designer. (The post-link RMO calls occur whether or not there are LINK paragraphs in the screen description (TRS).)

What if the screen designer wants the RMO to manipulate the key fields to be used by a LINK paragraph before the link is executed?

In order to meet the dual requirement that the RMO both be able to set the link keys and also that the RMO be able to use the link values, a **"pre-link"** RMO call is provided in addition to the **"post-link"** calls described above. (The pre-link RMO calls occur whether or not there are link paragraphs in the screen description (TRS).)

That is, the **RMO is called twice** at "BEGREC", and **twice** at each field entry. There is only one "EOFREC" call.

In order to distinguish the pre-link and post-link RMO calls, TRANS sets the second letter of M$M to 'X' in the pre-link call, as follows:[4]

```
"UX" if TRANS is in Update Mode
"AX" if TRANS is in Append Mode
"IX" if TRANS is in Insert Mode
"DX" if the active record is being deleted
```

4. The "DX" pre-link RMO call only occurs if the special field RJ$RJ is included in the local fields section of the RMS. See Section 16.1.2 “Reject APPEND, INSERT, UPDATE, DELETE, or Transfer”.

15.1.2.1 M$M_nn: Action Code For Button

An Action Code for a button can be specified as M$M_nn, where 'nn' is a number between '01' and '99', the same way M$M-nn can be specified in keyboard macros. For example:

```
Action=M$M_03
```

means that the RMO receives a call with M$M set to '03' when the BUTTON is pressed [if F$UNCKEY is present, the RMO also receives an 'FX' call with F$UNCKEY set to 'rmo' ('CT_L' if Physical)].

15.1.3 Local Fields in the RMO

The RMO operates on actual fields in the active record, and on any local fields declared in the RMO itself. (The RMO should not operate on virtual fields because virtual fields are calculated **after** the RMO is called.) A local field in the RMO can be displayed on the terminal, and can be changed by the user at the keyboard or the RMO. (See Section 9.5 "LOCAL Section" for a definition of the RMO local field concept.)

An RMO references a link field by creating a local field of the same name and type as the link field. For example, if a zip code table included the CITY and STATE fields, a LINK paragraph in a screen description might be as follows:

```
LINK ZIP.TAB
KC ZIP
L CITY
L STATE
END
```

The CITY and STATE fields could be referenced in the RMO as follows:

```
...
LOCAL
S$S/A6
M$M/A2
CITY/A20
STATE/A2
...
```

Local fields that are not link fields in the screen description, are **not** reinitialized as TRANS moves from record to record. Hence they can be used by the RMO to keep values from record to record, e.g., using the RMO to maintain a batch total.

Local fields from the RMO are set up in the field names section of the screen description (TRS) as either ER (editable and refreshable) or DR (display and refreshable). (See Section 5.5.1 "Display" and Section 5.5.2 "Editable"). In the case of fields which are neither in the master file nor declared in a link paragraph in the screen description, the local field type is appended to the field name with a slash. The field type is optional in the field names section for local fields from link files. This is because SCREEN knows the file definition of the link files from the LINK paragraph. If present, the field type is checked against the field type that SCREEN read from the link file.

The local fields from the RMO are usable in the screen description. For example, the following local fields in the RMO:

```
...
AMOUNT/D
DATE/DA
...
```

may be included in the field names section of the screen description as follows:

```
ER AMOUNT/D
DR DATE/DA
```

When the RMO alters a "DR" or "ER" field that is on the screen TRANS automatically refreshes (redisplays) the field value. Consequently, any field alterable by the RMO should be designated "ER" or "DR" in the field names section of the screen description regardless of whether it is a local field or an actual field.[5] That is, an actual field from the master file may be included in the field names section of the screen description as "DR" instead of "D", or "ER" instead of "E", if the RMO is going to alter the field, and it must be refreshed after RMO calls.

An RMO can check for entry errors, perhaps applying logic beyond the scope of the Check statement's Boolean expression, and then set a local error code field. Then the Check statement can display an error message contingent on the contents of this error code field. The text of the error message can be in the TRS, or the error message text can be linked in from an error message table based on the value of the error code field (see Section 16.19 "Using the RMO with Table Driven Error Messages").

Also the RMO is uniquely capable of setting values in actual fields in the active record (and in linked fields in linked records) as a result of computations performed on data in the active record (and its linked fields).

15.2 Order of Events

Before one can write an RMO to operate behind the screen, it is important to understand the order of events that occur during processing. Logic in the RMO should always be associated with a certain "status" and "mode". For example, processing that might be done at "BEGREC" status in "UP" mode would not be done at "SS#" status in "AP" mode. The order of events is different for each "status" and is now presented. Note carefully the points in the processing where the RMO is called, and the "status" and "mode" associated with each RMO call.

5. Actual fields from the master file may be designated "LR" to indicate that the field should be refreshed on the screen when the RMO changes it, and also that field value changes should be logged in the field log file. Changes to "LR" fields made by the RMO are logged only when LFEXIT control is active (see Section 6.3.1.1 "Update Mode Under LFEXIT Control").

15.2.1 Beginning of Record Processing: S$S = 'BEGREC'

The order of events that occur at the "beginning of record" (BEGREC) is as follows:

1. A record from the master file is read or in the case of Append Mode, the blank (null) record is created.
2. The pre-link RMO call is made. S$S contains 'BEGREC' and M$M contains the current TRANS mode using the pre-link code ('UX', 'AX', 'IX'). The RMO executes, perhaps altering fields to be used by a LINK paragraph.
3. All LINK paragraphs (see Section 5.4.1 "LINK Paragraph") in the screen description are executed.
4. Then the post-link RMO call is made. S$S still contains 'BEGREC' and M$M now contains the code for the current mode of TRANS without the pre-link "X" ('UP', 'AP', 'IN'). The RMO executes, perhaps altering actual fields in the record, or local fields, and returns to TRANS.
5. The Virtual fields (see Section 5.5.4 "Virtual Fields") and Message statements (see Section 5.5.7 "Message Fields") from the field names section are executed.
6. If the RMO set a condition-letter for an APPEND paragraph (see Section 5.4.2 "APPEND Paragraph"), the record is now appended (or inserted or deleted) to an external file.
7. The record is displayed and the cursor is positioned at the first editable field in the field names section of the screen description.

15.2.2 Field by Field Processing: S$S = 'fieldname'

RMO calls occur when a value is entered in a field. (A more advanced technique also causes an RMO call when a field is being **skipped** by the user. This is described in Section 16.4 "Controlling the Skipping of Fields: SK$SK". The discussion here deals with the routine RMO call.)

The order of events that occur when a value is entered into a field is as follows:

1. The user types the value, and presses the ENTER keystroke or an automatic carriage return occurs when the field is completely entered (see Section 5.3.1.3 "AUTOCR: Automatic Carriage Return").
2. The value entered is inspected for format, e.g., is an alphabetic character being entered into a numeric field. If there is a format violation, the value is rejected and TRANS is put into Error Mode. When the user presses the ERR keystroke to clear the Error Mode, the original value is redisplayed and the user can continue. If the value entered satisfies the format of the field type, the processing continues.
3. The pre-link RMO call is made. S$S contains the field name into which the value has just been entered, and M$M contains the current TRANS mode using the pre-link code ('UX', 'AX', or 'IX'). The RMO executes, perhaps setting up keys to be used by a LINK paragraph.
4. Any link(s) triggered by a manually entered "KC" or "C" field in a LINK paragraph is performed.
5. Then the post-link RMO call occurs. S$S still contains the field name into which the value has just been entered, and M$M now contains the code for the current mode of TRANS without the pre-link "X" designation ('UP', 'AP', or 'IN'). The RMO executes, perhaps altering fields in the record or local fields, and returns to TRANS.

6. The Virtual fields, Message statements, and the "C" Check statements (see Section 5.5 “Field Names”) from the field names section of the screen description are executed (but not displayed).
7. If the RMO set a condition-letter for an APPEND paragraph, the record is now appended (or inserted or deleted) to the append file.
8. If any "C" Check statement evaluates to true, i.e. an error is detected, then TRANS is put into Error Mode. When the user presses the ERR keystroke to clear the error, link fields set by the erroneous value are reset to their old value(s), and then the RMO is called with M$M set to "ER" and the field name still in S$S. When the RMO returns, the Virtual fields are re-evaluated, and the original (unaltered) value is redisplayed.
9. If all "C" Check statements evaluate to false, i.e., no error is detected, then the entered field, Virtual fields, Message fields, and all "ER", "DR" and "LR" fields altered by the RMO, are all redisplayed.
10. In Update Mode (without NOWRITE or LFEXIT control) the active record is written back to the disk.

15.2.3 End of Record Processing: S$S = 'EOFREC'

TRANS performs several functions just as the record on the screen is about to leave the screen and either be replaced by a new record or the current screen is to exit. These functions are called "end of record" processing.

There are a variety of ways for the user, or for the RMO logic to trigger the "end of record" condition. These are:

1. The NEXT keystroke. (User)
2. The NBRK keystroke. (User)
3. The PREV keystroke. (User)
4. The PBRK keystroke. (User)
5. The NREC keystroke. (User)
6. The EXIT keystroke. (User)
7. The BRNC keystroke followed by a branch code. (User)
8. The XRET keystroke. (User)
9. Entering key value(s) causing a record search in Update Mode. (User)
10. An automatic branch (B$B, see Section 16.2 “Automatic Branching: B$B and R$R”).
11. An automatic return from a branch (R$R, see Section 16.2 “Automatic Branching: B$B and R$R”).
12. Displaying the top of file record (F$F, see Section 16.7 “Top of File Control: F$F”).

(Note that pressing the APND keystroke while in Append Mode neither initiates end of record processing nor files the record currently being displayed.)

The order of events that occur at the "end of record" (EOFREC) is as follows:

1. Once end of record processing is initiated, if either of the following conditions occurs, TRANS goes into Error Mode, and the end of record (EOFREC) RMO call is **not** made.
   a. If TRANS is in 'UP' mode and LFEXIT control is active (see Section 6.3.1.1 "Update Mode Under LFEXIT Control"), or if TRANS is in 'AP' or 'IN' mode, and any CLF Check statements (see Section 5.5.6 "Check Statement") is evaluated as true, TRANS goes into Error Mode.
   b. If TRANS is in 'UP' mode and LFEXIT control is active, or if TRANS is in 'AP' or 'IN' mode, and any required fields (see Section 5.5.5 "REQUIRE Statement") from REQUIRE statements contain null values, TRANS goes into Error Mode.

   The user must press the ERR keystroke to clear the error condition and continue.
2. If all CLF Check statements are evaluated as false, the end of record RMO call occurs. S$S contains 'EOFREC' and M$M contains the terminal mode ('UP', 'AP', or 'IN').
3. If TRANS is in 'AP' or 'IN' mode, the RJ$RJ field (see Section 16.1.2 "Reject APPEND, INSERT, UPDATE, DELETE, or Transfer"), if present, is tested. If it is set to "1" TRANS enters Error Mode with the message "RECORD NOT ACCEPTED" and end of record processing stops.
4. If the RMO set a condition-letter for an APPEND paragraph (see Section 5.4.2 "APPEND Paragraph") the record is now appended, or inserted, or deleted to the append file.
5. All link writebacks are performed. The fields that can be written back are the link fields that appeared with an 'L' in a LINK paragraph, where the LINK paragraph had the 'W' for writeback present after the link file name on the LINK statement (see Section 5.4.1 "LINK Paragraph"). TRANS checks if any of the link field values have been changed. If TRANS finds that any link field value has been changed then that link record is written back to the disk.
6. All the INDEX paragraphs (see Section 5.4.3 "INDEX Paragraph") in the screen description are executed. That is, if any indexed fields have been altered in the screen, then the record in the index file which pointed to the active record based on these altered fields is replaced with an index record reflecting the new values.
7. In Append Mode or Insert Mode, or in Update Mode under LFEXIT control, the active record is written to disk.
8. If the end of record RMO call in event 2 above requested a second end of record RMO call after all records had been written to the disk, the second call is made (see Section 16.8 "Post-Writeback EOFREC RMO Call: B$OB").

NOTE

The Check statement (designated by the letter "C") is not evaluated at end of record processing.

15.2.4 Processing Record Deletions

The order of events that occur at record deletion is as follows:

1. TRANS receives the DEL keystroke (see Section 6.7 “Record Operations”). Immediately it checks if the current record is the last record in the file or the last record in a locked range (see Section 5.5.1.1 “Restrict TRANS to Key Range”). If this is the case the DEL keystroke is ignored.
2. If the special local RMO field RJ$RJ (see Section 16.1.2 “Reject APPEND, INSERT, UPDATE, DELETE, or Transfer”) is present, the RMO is called with M$M (mode) set to "DX".
3. If the RMO sets RJ$RJ to 1 at the "DX" RMO call TRANS echoes a "bell" character for the DEL keystroke, and exits the delete sequence of operations (i.e. delete is blocked.)
4. TRANS starts the delete verification dialogue (see Section 6.7 “Record Operations”). If the verification dialogue is not completed successfully TRANS will exit the delete sequence without deleting a record.
5. The RMO is called with M$M (mode) set to "DE".
6. The record is deleted.

15.2.5 Processing Record Transfers

The order of events that occur for record transfer (TRF keystroke) operations is as follows:

1. If the special local RMO field RJ$RJ (see Section 16.1.2 “Reject APPEND, INSERT, UPDATE, DELETE, or Transfer”) is present, the RMO is called with M$M (mode) set to "DX".
2. If the RMO sets RJ$RJ to 1 at the "DX" RMO call TRANS echoes a "bell" character for the TRF keystroke, and exits the transfer sequence of operations (i.e. the record transfer is blocked.)
3. TRANS starts the record transfer dialogue and processes the transfer as described in Section 6.7 “Record Operations”.

15.3 TRANS RMO Debug Facility

Section 9.11 “DEBUG Mode” describes how commands that call an RMO can run the RMO in **Debug Mode.** AdmTrans for Windows has a special “GUI”[6] version of Debug Mode, that has all the capabilities of the traditional debugger packaged in an easy-to-use “point and click” interface.

To use debug mode, the RMS must first be compiled with the **-ADBG** option:

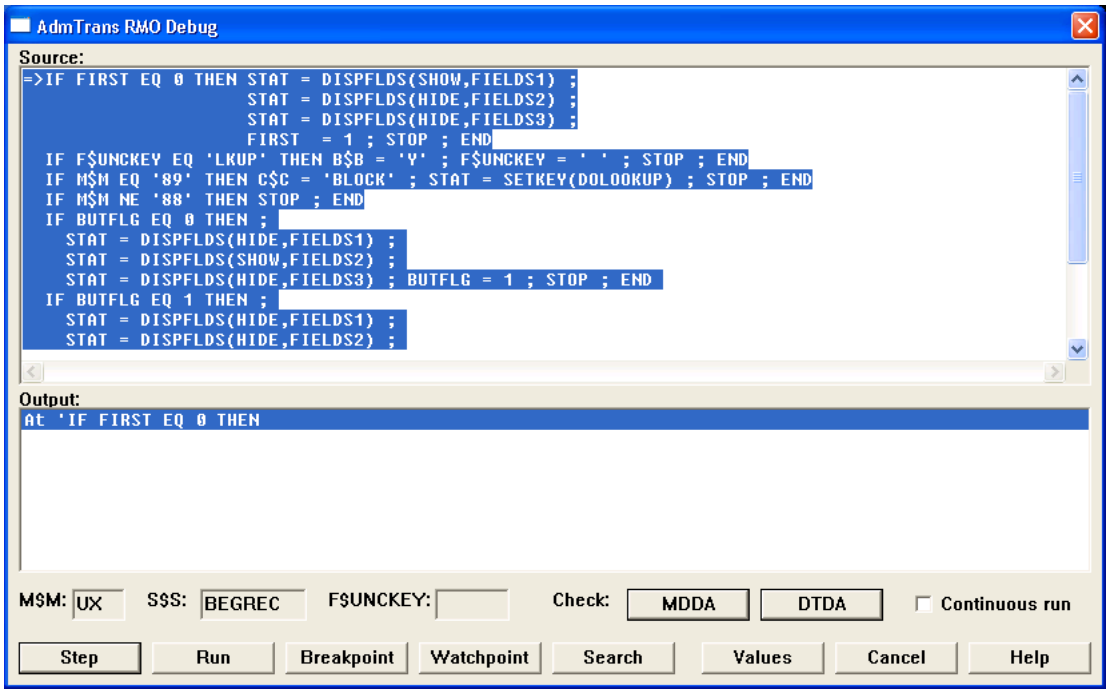
```
>admcmp rms_name -ADBG
```

6.Graphic User Interface

and the the logical name **ADM_DEBUG_RMO** must be assigned:

```
>admlcr adm_debug_rmo Y
```

Whenever the RMO is ready to be executed, the RMO Debug window will pop up, (at the beginning of the TRANS session it pops up before the TRANS screen display, because the RMO is called before TRANS creates the display).



The RMO Debug window contains:

1. The “Source” panel displays the RMO source code, with an indicator
   =>
   showing which line of code is about to be processed. You can mouse-click (and/or use the arrow keys) to place the text cursor in the source window (to indicate the location for a breakpoint).
2. The“Output” panel displays progess through the statements as they execute. Informational messages and “Search” results also display here.
3. The current values of the special RMO fields MM, SS, and F$UNCKEY.

4. An array of function buttons:

   click 

   to execute the next statement in the RMO

   click 

   to run the RMO up to the next Breakpoint, Watchpoint, or until the RMO call is completed.

click 

to set a breakpoint at the location of the text cursor in the "source" panel

click 

to open up a window showing all the available fields, allowing you to toggle watchpoints on any field

click 

to search the source code for a particular string, the results are displayed in the output window.

click 

to open up the values window, where you can click on any of the fields in the virtual record to see its value at the current point of the RMO execution

click 

to exit debug mode, TRANS returns to normal operation

click 

to display a window that displays Help for the TRANS RMO debugger.

The 

checkbox is a toggle that controls if the debugger will stop at the beginning of the RMO each time, or silently run till a breakpoint occurs.

Click on one of these two buttons

to have TRANS do an integrity check on its two internal arrays (meta-data and data). This will create two files, mdda.dmp and dtda.dmp, in the user's default directory. These files are for interpretation by ADMINS support staff.

15.4 Examples Using an RMO Behind the Screen

In order to illustrate the use of an RMO behind the screen, we now present several examples. Note that in all the examples the processing in the RMO is determined by the current "status" (S$S) and the current "mode" (M$M).

To review, the RMO is automatically called twice at the beginning of each record (BEGREC), twice after each field is entered, and once when the record is leaving the screen (EOFREC). The logic in the RMO should be written so that the RMO performs only the instructions necessary, based on the current mode (M$M) and status (S$S) set by TRANS. TRANS operates more efficiently when only the necessary instructions are executed in each RMO call. Typically, RMO logic is based on the post-link RMO call when the user types into a particular field. Furthermore, if the logic in the RMO is not structured to avoid redundant execution of RMO instructions, certain operations (e.g. arithmetic) could yield unexpected results because instructions were executed more times than the user intended.

15.4.1 Accounts Payable Example

The user is entering payments for invoices. The user enters a purchase order number, a check number, a vendor number, an invoice number and an amount. The screen procedure is expected to do the following.

1. Automatically insert the date into the payment record.
2. Display the vendor number, encumbered amount, and paid-to-date values from the PO file when the purchase order number is entered.
3. Display an error message if either the purchase order does not exist in the purchase order file, or if the vendor number doesn't match the vendor number in the purchase order file, or if the payment amount added to the paid-to-date for that purchase order exceeds the total encumbrance in the purchase order. (An encumbrance is a commitment related to unperformed contracts for goods and services.)
4. Display the vendor name and address from the vendor file, and if instructed to "pay" then the screen should add the payment to a the (temporary) paid-to-date field, and append the check number, vendor number, invoice number, date, and amount to the check file.
5. When the record is filed, the actual paid-to-date field is set to the temporary paid field, and is written back to the PO file.

The pertinent file definitions follow.

```
*     PAY.DEF     "payment file definition"
      MAS 1000
      DATE DA KEY1       "payment date"
      PO# X9999 KEY2     "purchase order number"
      VEND# X9999        "vendor number"
      INV# A10           "invoice number"
      AMT D2             "amount of payment"
      ...
      *     PO.DEF       "purchase order file definition"
      MAS 3000
      PO# X9999 KEY1     "purchase order number"
      VEND# X9999        "vendor number"
      ENCUMB D2          "encumbered amount"
      PAYTD D2           "paid to date"
      ...
      *     VEND.DEF     "vendor file definition"
      TAB 4000
      VEND# X9999 KEY1   "vendor number"
      VENDOR A30         "vendor name"
      ADDR A30           "vendor address"
      CITYST A30         "vendor city state"
      *     CHECKS.DEF   "check file definition"
      MAS 500
      CK# I  KEY1        "check number"
      VEND# X9999        "vendor number"
      INV# A10           "invoice number"
      DATE DA            "payment date"
      AMT D2             "amount of payment"
```

The screen description could look as follows:

```
PAY PAY.MAS 1 PAY.RMO NOMSG APPEND
*
*  link paragraph for purchase order file
*
LINK PO.MAS W
KC PO#
*  linked VEND# is renamed PVEND in the screen
L  VEND# PVEND
L  ENCUMB
L  PAYTD
END
*
*  link paragraph for vendor file
*
LINK VEND.TAB
KC VEND#
L  VENDOR
L  ADDR
L  CITYST
END
*
*  append paragraph for checks
*
APPEND CHECKS.MAS ACT P
CK#
VEND#
INV#
DATE
AMT
END
*
*  field names section
*
E DATE
E PO#
ER CK#/I
E VEND#
E INV#
E AMT
```

```
ER ACT/A1 [9,16,1]
*
*       fields linked from PO.MAS
D PVEND
D ENCUMB
*       changed by PAY.RMS and written back
DR PAYTD/D2
*        a local field temporarily holds the paid amount
DR PAID/D2
*
*       fields linked from VEND.TAB
D VENDOR
D ADDR
D CITYST
*
* TODAY must be in the TRS to be used in the RMS
DR TODAY/DA
*
*  Error Messages
*  Note, each Check statement only applies when the pertinent
*  PAY.MAS field is non-zero.
*
C PO# NE 0 AND PVEND EQ 0000 AND ENCUMB EQ 0
PURCHASE ORDER NOT IN PO FILE
*
C VEND# NE 0000 AND #VEND NE PVEND
VENDOR # ENTERED DOESN'T MATCH PO FILE
*
C AMT NE 0 AND AMT + PAYTD GT ENCUMB
PAY AMOUNT EXCEEDS PURCHASE ORDER AMOUNT
*
*     screen layout section
*
SCREEN
CE PAYMENT ENTRY SCREEN
BL
   DATE: DATE-----         PO: PO#-
BL
     CK: CK#--          VENDOR: VEND-
BL
INVOICE: INV#------    AMOUNT: ---------AMT
BL
TYPE P FOR PAY:
BL
CE INFORMATION FROM PURCHASE ORDER
BL
VENDOR: PVE-  ENCUMBERED: ------ENC  PAID TO DATE: -----PAID
BL
CE VENDOR INFORMATION
BL
VENDOR------------------------
ADDR--------------------------
CITY--------------------------
END
```

The record maintenance procedure (PAY.RMS) might look as follows.

```
FILE PAY.MAS
LOCAL
S$S/A6
M$M/A2
TODAY/DA
ACT/A1
PAYTD/D2
*       PAID is a local field which temporarily holds the
*       PAYTD amount until the record is filed, and allows
*       the payment amount to refresh on the screen.
PAID/D2
PROGRAM
*
* Only execute the RMO in 'AP' mode.
*
IF M$M NE 'AP' THEN GOTO OUT END
*
* At 'BEGREC' fill in the date.
*
IF S$S EQ 'BEGREC' THEN DATE = TODAY ; GOTO OUT END
*
* When PO# is entered, set the PAID field equal to
* the linked value of PAYTD from the PO file.
*
IF S$S EQ 'PO#' THEN PAID = PAYTD END
*
* The next statement adds the payment to PAID.
* Adding is triggered by the same condition, i.e.
* putting a "P" in ACT, that triggers appending to the
* check file.
*
IF S$S EQ 'ACT' AND ACT EQ 'P' THEN PAID = PAYTD + AMT END
*
* At 'EOFREC' the PAYTD field is set to the value of the
* PAID field. PAID is initialized for the next transaction.
*
IF S$S EQ 'EOFREC' THEN PAYTD = PAID ; PAID = 0 END
*
OUT: STOP
```

15.4.2 Example of Appending Via the RMO

Section 5.4.2 “APPEND Paragraph” describes how to build and add records to an external file by setting a letter into the condition-name field specified in the Append Paragraph. It is also possible to let the RMO behind the screen set the letter and cause the "append" record to be built and added. This is done simply by including the condition-name field as a local field in the RMO. (The condition name field should then be a 'DR' field on the screen as described in Section 15.1.3 “Local Fields in the RMO”.)

Appending via the RMO may be used to build logs specific to the application to supplement or replace the ADMINS field log. Any user action may be logged, not just changes to records. Appending via the RMO allows the application designer to have more control over when a record is appended.

For example, we may have a telephone lookup application where a name is entered and the telephone number is displayed. We wish to create a usage log showing name, time and date for each query.

```
*     DIREC.DEF
*
MAS 20000
NAME A40 KEY1      "name"
TELEPHONE A12      "telephone number"
TITLE A20          "title"

DATE DA KEY1       "date of query"
TIME A8 KEY2       "time of query"
NAME A40           "name queried"     *      LOG.DEF
*
MAS 10000


*      DIREC.TRS
DIREC DIREC.MAS 1 DIREC.RMO NOMSG
*
* Append Paragraph to append to LOG.MAS
*
APPEND LOG.MAS LOGIT A
DATE
TIME
NAME
END
*
*
* Note that TODAY and NOW must be in the TRS
* in they are to be used in the RMS
*
DR TODAY/DA
DR NOW/A8
E NAME
D TITLE
D TELEPHONE
DR DATE/DA
DR TIME/A8
DR LOGIT/A1
SCREEN
CE ENTER NAME       TODAY----     NOW-----
BL
NAME: NAME----------------------- TITLE: TITLE--------------
BL
TELEPHONE: TEL---------
END

*      DIREC.RMS
*
FILE DIREC.MAS
LOCAL
S$S/A6
M$M/A2
TODAY/DA
NOW/A8
DATE/DA
TIME/A8
LOGIT/A1
PROGRAM
DATE = TODAY ; TIME = NOW
IF S$S EQ 'NAME' AND M$M EQ 'UP' THEN LOGIT = 'K' END
```

15.4.3 Example Using Global Fields

In Section 5.5.9 "Global Fields" we described the "global field" facility. Global fields represent a portion of memory in TRANS that is not erased as the user branches from screen to screen. By storing information in the global field area, the information is then available to each screen called until the user exits TRANS.

Remember, the **definition** of the global fields, **and the order** in which they are presented, **must be the same** in each screen description (TRS) where they are used in common[7]. (Only the global fields which are actually referenced need to be included in the RMO behind the screen. The order of the global fields in the RMS does not matter.) The use of global fields is best shown in an example.

We wish to create a "logon" screen where each user identifies herself/himself, enters a password, and is then passed on to the first application screen. The user can to be restricted to a particular terminal number. (See Section 5.5.8.3 "Terminal Number" for a description of the internal field T$T.) We will keep the user-id, and the time logged on, as global fields. The CONTROL.MAS file will hold the terminal number, password and branch code for the first application screen for each user-id. (The reader may wish to examine Section 16.2 "Automatic Branching: B$B and R$R" in order to understand the automatic branching used in the following example.) LOGON.MAS is a dummy file containing one record with a blank value in the key field 1DUMMY.

```
*      LOGON.DEF
MAS 100
1DUMMY A2 KEY1      "dummy key field"
2DUMMY A2           "another dummy field"

*      CONTROL.DEF
MAS 100
USERID A8 KEY1    "user's identification"
PASSWORD A8       "user's password"
1STBRANCH A2      "user's first branch code"
TERMINAL# A4      "user's terminal number"
```

The definitions for the global fields are as follows:

```
G$WHO/A8      "USERID of the current user"
G$TIME/A8     "time the current user started"
```

Remember, global fields act as if there is a one record file that remains in memory until the user exits TRANS. These fields are included in all the screen description and record maintenance procedures that will use the global area.

7. The easiest way to keep global fields in order is to use the same STRUCTURE paragraph in every screen, as described in Section 5.5.9.1 "STRUCTURE: Lay out global fields section".

The screen description follows:

```
*      LOGON.TRS
LOGON LOGON.MAS 1 LOGON.RMO NOMSG NOBR
*
* Link a user-id to control file
*
LINK CONTROL.MAS
KC USERID
L PASSWORD
L 1STBRANCH
L TERMINAL#
END
*
*              global field definitions
DR G$WHO/A8      "USERID of the current user"
DR G$TIME/A8     "time the current user started"
*
ER USERID/A8
ER PASSW/A8
DR TODAY/DA
DR NOW/A8
DR T$T/A4
E 1DUMMY
*
C PASSW NE ' ' AND PASSWORD
UNAUTHORIZED USERID/PASSWORD
C PASSWORD NE PASSW AND T$T NE TERMINAL#
INCORRECT TERMINAL
*
SCREEN
CE LOGON SCREEN   TODAY---- NOW------
BL
1D-
       TYPE USER ID: USERID--
BL
        TYPE PASSWORD: PASSW-----
        PRESS NEXT
BRANCHES
A ...
...
B ...
...
C ...
...
END
```

```
*     LOGON.RMS
FILE LOGON.MAS
LOCAL
*        global field definitions
G$WHO/A8
G$TIME/A8
*
S$S/A6
M$M/A2
B$B/A2
PASSW/A8
1STBRANCH/A2
NOW/A8
PROGRAM
IF S$S EQ 'EOFREC' AND PASSW NE ' ' THEN
  B$B = 1STBRANCH ; G$WHO = USERID ;
  G$TIME = NOW END
```

15.4.4 Example Using a Pre-Link RMO Call

A screen is used to enter payments into a payment file. The bill number is a 12 digit number that contains the account number as well as a batch control number. The account number is contained in digits 3 to 8 of the bill number. The account number is used to link to an accounts receivable file where the screen checks that the customer account exists and also applies the payment.

```
*     PAYM.DEF
*
MAS 10000
SEQ I KEY1             "payment sequence number"
BILL# X999999999999    "bill number"
AMT D2                 "payment amount"

*     ACCT.DEF
*
MAS 20000
ACCT# X999999 KEY1     "account number"
NAME A30               "customer name"
BALANCE D2             "balance due"

*     PAYM.TRS
*
PAYM PAYM.MAS 1 PAYM.RMO APPEND NOMSG
*
*   Link to the Accounts Receivable file.
*   The writeback is for applying the payment (AMT)
*   to BALANCE.
*
LINK ACCT.MAS W
KC ACCT#         "the key field in the link is a local field"
C BILL#
L NAME
L BALANCE
END
*
E SEQ
E BILL#
DR ACCT#/X999999
D BALANCE
D NAME
E AMT
*
C ACCT# NE '000000' AND NAME EQ ' '
INVALID ACCOUNT IN BILL NUMBER
*
SCREEN
CE ENTER PAYMENTS
BL
SEQ: SE-   BILL#: BILL#-------      ACCOUNT: ACCT#-
BL
           NAME: NAME-------------------------
           AMOUNT: -------AMT
END
```

The first statement in PAYM.RMS below says that during the 'BEGREC' pre-link call in Append Mode, set the value in the ACCT# local field to 0.

The second statement in PAYM.RMS below says that when we get a pre-link call in Append Mode just when the BILL# has been entered, then construct the account number (ACCT#) using the 3rd through the 8th digit of BILL#, using the STR subroutine (see Appendix H.5.1 “STR - Select Part of a Field”.).

The third statement of PAYM.RMS says that when the AMT field is entered in Append Mode at the ("normal") post-link call then apply the payment in AMT to the linked in BALANCE from ACCT.MAS. The new BALANCE will be written back via the writeback (W) in the LINK statement of the TRS.

The fourth statement in PAYM.RMS says that if there is an Error Mode call after BILL# has been entered, i.e. if the ACCT# was not found in ACCT.MAS, then reset ACCT# to zero. This is done because the Check statement in PAYM.TRS uses the presence of a non-zero value in ACCT# to indicate that the link to ACCT.MAS failed to find the account number there.

```
*     PAYM.RMS
*
FILE PAYM.MAS
LOCAL
S$S/A6
M$M/A2
BALANCE/D2
ACCT#/X999999
PROGRAM
*
* At 'BEGREC' in Append Mode set the ACCT# local field to 0.
*
IF S$S EQ 'BEGREC' AND M$M EQ 'AX' THEN ACCT# = 0 END
*
* When the BILL# is entered, string out the 3rd - 8th
* characters and put them into the ACCT# field.
*
IF M$M EQ 'AX' AND S$S EQ 'BILL#' THEN ;
  ACCT# = STR(ACCT#,BILL#,'3/I','8/I') ; GOTO DONE END
*
* When the AMT is entered, subtract the AMT from the BALANCE
*
IF M$M EQ 'AP' AND S$S EQ 'AMT' THEN ;
  BALANCE = BALANCE - AMT ; GOTO DONE END
*
* If an error occurs when entering the BILL#, reset the
* ACCT# back to 0.
*
IF M$M EQ 'ER' AND S$S EQ 'BILL#' THEN ACCT# = 0 END
DONE: STOP
```

Chapter 16:Advanced RMO Functions with TRANS

Manual Chapter 15: "Basic RMO Functions with TRANS" describes the general purpose of an RMO behind the screen, the communication between the screen and the RMO, and includes several general examples.

This section describes the specific functions which may be included in an RMO behind the screen. The purpose of many of these functions is to allow more tightly controlled screens to be developed, to provide more user-friendly screens and more thorough data entry validation. Some of the advanced RMO functions include: controlling the writeback to disk, rejecting new or changed records due to an error condition, automatic branching, automatic record insertion, cursor control, highlighting fields, printing messages, timing out TRANS, record selection, and special subroutines which operate only in an RMO behind the screen.

16.1 Controlling Changes Written To Disk

In Update Mode, the only time TRANS writes the active record back to the disk is when the user enters a manual change.[1] Therefore an RMO behind the screen could alter the record at "BEGREC", "EOFREC", or after the last manual change was made, and the changes would not be written back to the disk. Also, the writeback to link files is done **only** as part of the end of record processing ("EOFREC").

The user, if necessary, may control both the writing of the active record during Update Mode, and the performance of the link writebacks. This is done by setting a local integer field in the RMO called "W$W" to one of the following values: (W$W used in conjunction with the keyword NOWRITE in the screen header line handles writing records differently in Update Mode, and is described in Section 16.1.1 "High Volume Update: NOWRITE".)

0	When the local field W$W remains "0", which is its initial setting, the normal writing of records is done. That is, the active record is written after each manually entered field change and the link writebacks are done at "EOFREC".

1. Writing back to disk in Update Mode under LFEXIT control is described in Section 6.3.1.1 "Update Mode Under LFEXIT Control". This section describes when TRANS writes back the active record to disk in Update Mode without LFEXIT control.

1	In single record screens program the RMO to set W$W to “1” whenever TRANS is to write the active record back to the disk. This may be done at “BEGREC”, “EOFREC”, or at any time the RMO is called. In this way, the RMO can cause the active record to be written to the disk even though the user did not manually change a field. TRANS writes the active record (at the next post-link call) and automatically resets W$W to “0”. In multi-record screens setting W$W to 1 has no effect. Multi-record screens always write the active record as each field is entered.
2	The RMO setting W$W to “2” at “EOFREC” instructs TRANS to skip writing the active record and also skip the writeback of the LINK records. Again, W$W is automatically reset to “0”. In multi-record screens setting W$W to 2 only effects LINKs. Multi-record screens always write the active record as each field is entered.
3	Normally link writebacks are performed only at "EOFREC". By setting W$W to "3" at any RMO call, TRANS will writeback **all** LINK records at that point in the processing (i.e. the next post-link call). W$W is automatically reset to "0". Note that in multi-record screens, neither LINK W fields nor fields in the main file of the screen can be changed at BEGREC calls, because the records are not locked at that time. (Changes may be made at the MULREC call as described in Section 16.22 “Multi-Record RMO Support”).
4	Combines functions of W$W = 1 and W$W = 3, i.e. when W$W is set to "4" at any RMO call both the active record is written and all link writebacks occur (at the next post-link call). The value of W$W is reset to "0". For multi-record screens this setting will act the same as W$W = 3.
5	When W$W is set to "5" at any **pre-link** RMO call TRANS will writeback **all** LINK records at that point in the processing. This setting forces write-back to the "old" LINK records **before** TRANS links to the new records (perhaps with values set at the "pre-link" call). After writing back the link records, W$W is automatically reset to "0". In multi-record screens setting W$W to 5 has no effect.

NOTE

W$W controls write-back during Update Mode only. If W$W is set in Insert Mode or Append Mode (IX, IN, AX, AP) the W$W setting is ignored and W$W is reset to zero.

W$W can be used to "delete mark" records. If W$W is set to 1 or 4 at the "delete" RMO call (see Section 15.2.4 “Processing Record Deletions”) the record can be written immediately before it is deleted. This technique can be used to mark records that have been deleted, should a subsequent FILECONVERT "sequentialize" operation (see Section 13.4.1 “Sequentialize an ADMINS data file”) cause the deleted records to re-appear.

16.1.1 High Volume Update: NOWRITE

Ordinarily Update Mode writes the record back to the disk each time a value is entered into a field. For high volume on-line data entry, Append Mode is more efficient than Update Mode because in Append Mode the record is written back to disk only when the NEXT keystroke is pressed to signal completion of the record. However, Append Mode has two disadvantages, particularly in high volume, multiple terminal applications. One, branching is not permitted in Append Mode, and, two, multiple users appending records to the same file have the added overhead of keeping track of each other's additions (see Chapter 19: "Concurrency Control: Multi-User Files").

Update Mode overcomes both these limitations, i.e., one can branch in Update Mode, and the number of records is constant. However, Update Mode rewrites the record as each field is entered or changed, which can cause excessive disk activity in high volume data entry applications.

If the keyword "NOWRITE" is placed on the screen header line then Update Mode will **not** write the active record back to the disk after each field change when W$W is "0". The active record is written back to the disk **only** when the RMO sets the W$W field to "1", which it should do at the "EOFREC" call only. Setting W$W to "2" or "3" performs the same functions as described in Section 16.1 "Controlling Changes Written To Disk" above. Setting W$W to "4" with NOWRITE combines the functions of W$W=1 and W$W=3, i.e. the LINKs are written back immediately, as with W$W=3, and the active record is written at "EOFREC" (W$W=1 with NOWRITE).

Hence, Update Mode with the "NOWRITE" option can be used for efficient high volume data entry into pre-allocated empty records. The "pre-allocated" file can be set up in such a way that each user could be updating (i.e. inputting) a different set of records in the file (e.g. a "batch"), and thus several user could be updating the same file simultaneously without locking each other out of a record (see Chapter 19: "Concurrency Control: Multi-User Files").

A alternative to using NOWRITE for more efficient data entry in Update Mode, is to use LFEXIT control (see Section 5.3.1.18 "LFEXIT or LFBACK: Update Mode Control" and Section 6.3.1.1 "Update Mode Under LFEXIT Control"). Under LFEXIT control, the active record is written to the disk only when the user presses the NEXT keystroke to file the record. Using LFEXIT control, an RMO is not needed to instruct TRANS to write the active record back. Once LFEXIT control is activated by typing into a non-key field, branching is permitted only after the record is filed via NEXT.

16.1.2 Reject APPEND, INSERT, UPDATE, DELETE, or Transfer

In Append Mode or Insert Mode[2] the active record is written to disk when the user presses the NEXT keystroke. The RMO behind the screen can refuse to accept a record being appended or inserted or updated, in Append ("AP") Mode or Insert ("IN") Mode. This is done by setting the local integer field RJ$RJ ("reject") to a non-zero value at "EOFREC". The NEXT keystroke will not file the record and the error condition "RECORD NOT ACCEPTED" is displayed. The user must press the ERR (\) keystroke to clear the error and continue. RJ$RJ can be used to prevent filing of the new record at the NEXT keystroke until the offending error(s) is (are) corrected.

RJ$RJ may be used in Update Mode to reject data entry only when LFEXIT control (see Section 6.3.1.1 "Update Mode Under LFEXIT Control") is active. In Update Mode under LFEXIT control, data entry is not written back to the disk until the user presses the NEXT keystroke. If RJ$RJ is set to a non-zero value at the "EOFREC" RMO call, the error message "RECORD NOT ACCEPTED" is displayed and none of the changes to the active record are written to disk. Although RJ$RJ can be used in reject data entry in Update Mode, the CLF Check statement (see Section 5.5.6 "Check Statement") is preferable for two reasons. First, CLF provides a facility for sophisticated error checking with little or no RMO logic. Second, each CLF error condition produces a specific Check statement message; whereas setting RJ$RJ produces a generic "RECORD NOT ACCEPTED" message which does not inform the user what is wrong with the record.

In the case of Update Mode without LFEXIT control, the record is re-written to the disk as each field is entered if there is no error generated via the format check on the entry or the Check Statement from the screen description. RJ$RJ does not have any effect in Update Mode when LFEXIT control is not active. In Update Mode without LFEXIT control, usually the Check Statement logic, perhaps assisted by additional error detection logic in an RMO operating behind the screen, is sufficient to validate update entries. There are occasions, however, where the data fields to be updated are so dependent on each other that some values must be accepted in Update Mode (and therefore written back to the disk) before a complete checkout can be performed. The NOWRITE option described in Section 16.1.1 "High Volume Update: NOWRITE" can be used to prevent writing the active record until all interrelated fields are entered and checked.

In Delete Mode the record is deleted when the user retypes the key fields after pressing the DEL keystroke (see Section 6.7 "Record Operations"). The RMO behind the screen can refuse to allow the record to be deleted. When the user presses the DEL keystroke and the field RJ$RJ is included in the RMO, the RMO is called with M$M (mode) set to "DX". The "DX" RMO call only occurs when RJ$RJ is included in the local fields section of the RMO. If the RMO sets RJ$RJ to "1", TRANS echoes a "bell" to the DEL keystroke and does not continue the delete sequence of operations, i.e. the deletion is blocked.

2. In the case of insert, a blank record with only the key value(s) set, is created immediately on the disk after the "ENTER I TO INSERT" prompt receives a positive response.

In record transfer (TRF keystroke) operations (see Section 6.7 "Record Operations") the record is transferred after the record transfer dialogue is successfully completed. The RMO behind the screen can refuse to allow the record to be transferred. When the user presses the TRF keystroke and the field RJ$RJ is included in the RMO, the RMO is called with M$M (mode) set to "DX". The "DX" RMO call only occurs when RJ$RJ is included in the local fields section of the RMO. If the RMO sets RJ$RJ to "1", TRANS echoes a "bell" to the TRF keystroke and does not continue the record transfer sequence of operations, i.e. the transfer is blocked.

16.1.2.1 Example of Using RJ$RJ

The following example shows the use of RJ$RJ to indicate an error at the end-of-record processing. When used in Append or Insert Modes the erroneous record will not be filed.

The example concerns the entry of tax payment records. The clerk enters a bill number, an account number, and then amounts for principal, interest and penalty. The screen links to an account file where the total amount owed is recorded. The RJ$RJ is concerned with checking that the sum of the three entered amounts, principal, interest and penalty, sum to the total owed in the account file.

```
BILL.DEF
     *
     MAS 10000
     BILL# X99999  KEY1 "tax bill number"
     ACCT# XA99999      "account number"
     PRINC D2           "principal amount"
     INT D2             "interest amount"
     PENALTY D2         "penalty amount"

ACCT.DEF
     *
     MAS 30000
     ACCT# XA99999 KEY1 "account number"
     TOTAL D2           "total amount owed"

BILL.RMS
     *
     FILE BILL.MAS
     LOCAL
     S$S/A6
     M$M/A2
     RJ$RJ/I
     TOTAL/D2
     PROGRAM
     RJ$RJ = 0
     *
     * At end of record compare the payment values to the
     * linked in total. Only accept an exact match.
     *
     IF S$S EQ 'EOFREC' AND PRINC + INT + PENALTY NE TOTAL
       THEN RJ$RJ = 1 END
```

```
BILL.TRS
     *
     BILL BILL.MAS 1 BILL.RMO APPEND INSERT NOMSG
     *
     * LINK TO ACCT.MAS FOR TOTAL OWED
     *
     LINK ACCT.MAS
     KC ACCT#
     L TOTAL
     END
     *
     *
     E BILL#
     E ACCT#
     *
     * Make sure we have the account on file
     *
     C ACCT# NE 0 AND TOTAL EQ 0
     ACCT# NOT ON FILE
     E PRINC
     E INT
     E PENALTY
     D TOTAL
     SCREEN
     CE ENTER TAX PAYMENTS
     BL
     BILL#: BILL-     ACCOUNT NUMBER: ACCT--
     ---------------------- PAYMENTS --------------------------
     PRINCIPAL: ------PRIN INTEREST: -------INT PENALTY: -----PENA
     BL
     PAYMENT MUST EQUAL TOTAL DUE: --------TOT
     END
```

16.2 Automatic Branching: B$B and R$R

Automatic branching allows the RMO behind the screen to invoke a branch.

To activate a branch the RMO sets the local field B$B to the branch code. (The branch code is the character the user would press after pressing the BRNC keystroke in order to manually perform the branch.) The branch is performed at the next point at which an operator-initiated branch would have been accepted and performed.

Similarly, setting the local field R$R to a non-blank character simulates an XRET keystroke, i.e., a return from last branch.

Setting B$B to "H" invokes TRANS HELP if it is available (if it is not available TRANS will attempt to branch to branch name "H").

The types of these local fields are B$B/A2 and R$R/A2.

Automatic branching is a very powerful technique that has many uses. Some of these uses are outlined here.

1. To control the switching from screen to screen.

   Manual branching, described in Section 5.7 “Branches”, creates a screen environment where the end-user at the terminal decides when and where to switch screens.

   An alternative is to have the logic in the RMO instruct TRANS when and where to branch. Sometimes this use of automatic branching can be used along with manual branching - where both the end-user at the terminal and the screen logic initiate branches. Or, using the screen header line keyword that inhibits manual

branching (NOBR), described in Section 5.3.1.14 “NOBR: Inhibit Manual Branching”, the end-user can be restricted exclusively to automatically generated branches. This latter, more controlled, screen environment is common in inquiry applications in order to implement security controls and also in complicated clerical data entry applications in order to guide the user through the data entry steps.

2. To fit more complex screen logic into TRANS.

   TRANS is designed to operate on one "physical master file" at a time, with a number of external files in relation to the physical master file -- links, indexes, logs, append files. The RMO operates on the logical master file consisting of the physical master file and the link and append files in relation. Some application functions may require the ability to have access to more than one logical master file, i.e. to use multiple screens with different physical master files.

   Via automatic branching (and other techniques such as Global Fields, see Section 5.5.9 “Global Fields”) the application designer can interrelate any number of screens to create an applications environment in which the end-user is only slightly aware that automatic branches are being invoked. Yet automatic branching provides the open ended ability to operate on multiple logical master files.

3. The application designer can present a set of choices as a menu on the terminal.

   The end-user can enter data (e.g. an account number) and, if necessary, a choice code (or the choice may be implied by the format of the account number), and then the RMO can automatically branch to the appropriate set of screens. In some applications **different users** could be presented with **different menus** after they enter their user identification which is linked to a file containing the operations they are permitted to perform (see example in Section 15.4.3 “Example Using Global Fields”).

4. To let the RMO choose from a very large number of branch choices.

   Section 5.7.3 “Calculated Branches” describes the method of using a single branch code and including the screen name of the branch to be taken in the field "B$fieldname". The RMO behind the screen may "calculate" a branch, i.e. set "B$fieldname", set B$B to the branch code, and automatically execute the branch. (This technique could also be used with manual branching.)

16.2.1 Example of Automatic Branching

A screen is being used to enter tax payments. The user enters a BILL# which calls up a record in the bill file. Then the screen allows the user to enter the payment amount. However, if the account number present on the bill record is also active in the arrears file with unpaid back taxes then the screen automatically branches to another screen that shows the arrears record in full detail, and allows the user to apply the payment there.

```
*    BILL.DEF
*
MAS 20000
BILL# X99999 KEY1     "bill number"
ACCT# XA9999          "account number"
AMTDUE D2             "amount due"
AMTPAID D2            "amount paid"
PAIDATE DA            "date of payment entry"

*   ARREARS.DEF
*
MAS 5000
ACCT# XA9999 KEY1     "account number"
TAX84 D2              "1984 tax owed"
TAX83 D2              "1983 tax owed"
TAX82 D2              "1982 tax owed"
ARPAID D2             "arrears already paid"

*   BILL.TRS
*
BILL BILL.MAS 1 BILL.RMO NOMSG MATCH
*
* link to ARREARS.MAS to see if there are back taxes
*
LINK ARREARS.MAS
KC ACCT#
L TAX84
L TAX83
L TAX82
L ARPAID
END
*
E BILL#
E ACCT#
D AMTDUE
D AMTPAID
DR TODAY/DA
DR NOW/A8
*
```

```
*screen is only 8 lines long
*
SCREEN 1 1 8 80
CE ENTER TAX PAYMENTS      TODAY----   NOW-----
BL
BILL#: BILL-     ACCOUNT NUMBER: ACCT-
BL
AMOUNT DUE: --AMTDUE        AMOUNT PAID: --AMTPAID
BRANCHES
A ARREARS ACCT#
EXAMINE ACCOUNT IN ARREARS
END
*


* ARREARS SCREEN
*
ARREARS ARREARS.MAS 1 NOMSG
E ACCT#
D TAX84
D TAX83
D TAX82
E ARPAID
SCREEN 9 1 8 80      "place arrears screen under bill screen"
CE ACCOUNT IN ARREARS  ACCT-
BL
             TAX DUE       ARREARS PAID: ---ARPAID
  1982    -----TAX82
  1983    -----TAX83
  1984    -----TAX84
END
```

The first statement in BILL.RMS below instructs TRANS that the rest of the RMO should be executed only during the post-link Update Mode RMO calls.

The second statement in BILL.RMS says that if there are back taxes owed then automatically branch to the arrears screen. Note, that if the account number in the bill file is not in the arrears file then all the link fields (i.e. TAX82, TAX83, TAX84, ARPAID) are set to zero by the LINK paragraph itself.

The third statement in BILL.RMS sets the PAIDATE (date payment entered) field to the current date when the AMTPAID field is entered.

```
*   BILL.RMS
*
FILE BILL.MAS
LOCAL
S$S/A6
M$M/A2
B$B/A2
TAX82/D2
TAX83/D2
TAX84/D2
ARPAID/D2
TODAY/DA
PROGRAM
IF M$M NE 'UP' THEN STOP ; END
IF S$S EQ 'BEGREC' AND TAX82 + TAX83 + TAX84 GT ARPAID
  THEN B$B = 'A' END
IF S$S EQ 'AMTPAID' THEN PAIDATE = TODAY END
```

The return to the BILL screen is via the XRET keystroke. Another RMO could be written to operate behind the ARREARS screen to automatically perform the return to the BILL screen.

16.2.2 Automatic NEXT key: B$B = 'LF'

TRANS acts as if the NEXT keystroke has been pressed if the RMO sets B$B to "LF". This feature has many applications in more advanced transaction screen applications. For example:

1. Use the automatic NEXT key to cause the record to be written to the disk in Append Mode, Insert Mode, or Update Mode when LFEXIT control is active (see Section 6.3.1.1 “Update Mode Under LFEXIT Control”), rather than requiring the user to press the NEXT key.
2. Complex screen processes that access multiple records can be implemented automatically via index files, branching and automatic NEXTs.

16.2.3 Automatic PREV Keystroke: B$B = 'BS'

It is possible for the RMO "behind the screen" in TRANS to invoke the PREV keystroke. This is done by setting the local field B$B to "BS" (for "backspace"). This feature allows the application developer to automatically, under control of the RMO running behind the screen, move TRANS from the currently active record to the previous record in the file, i.e. to "backspace" one record in the file. In the case of multi-record screens setting B$B to "BS" will caused TRANS to go back to the previous control break, just as it would if the PREV keystroke had been used. Whenever the local field B$B is set to "BS" by the RMO TRANS performs the PREV function and then resets the value of B$B to blank.

16.2.4 Automatic Exit From TRANS: B$B = 'CB'

It is possible for the RMO behind the screen to cause TRANS to exit. This is done by setting the local field B$B to "CB". This feature causes TRANS to act as if the user pressed the EXIT keystroke followed by a RETURN to return to the operating system prompt.

The automatic exit always functions, even if a manual exit from the screen is blocked by the keyword "NOEX" (see Section 5.3.1.16 “NOEX: Inhibit Screen Exit”) on the screen header line.

The example showing the use of the time-out facility (see Section 5.5.20 “TIMEOUT statement”) includes the use of the automatic exit.

16.2.5 Automatic Insert: B$B = 'IN'

rIt is possible for the RMO behind the screen to cause a record to be inserted into the master file without operator intervention. If the key fields of a record are set at "BEGREC" in the RMO behind the screen and B$B is set to "IN", then TRANS will do a record search and if a record with the set key values does not exist, then TRANS will insert the record. The cursor is placed at the first editable field and the terminal is placed in Insert Mode. If the record already exists, the record is displayed and TRANS is placed in Update Mode.

16.2.5.1 Automatic Insert Example

Since automatic insertion is requested at the "BEGREC" RMO call, an additional field is needed to indicate that the record insertion was requested. In the following example, the screen is called as a result of a branch with the desired key value in a global field. The only time the automatic insert is tried is at the first "BEGREC" call following the branch and in the example below, this is controlled by the local field SW.

```
...
G$REC/I
M$M/A2
S$S/A6
B$B/A2
SW/I
PROGRAM
IF S$S EQ 'BEGREC' AND SW EQ 0 THEN REC = G$REC ;
 B$B = 'IN' ; SW = 1 END
```

16.2.6 Bookmarking Screen, Returning to a Bookmarked Screen

Bookmarking is implemented by having AdmTrans store an extra copy of its internal 'branch history' (see Section 6.9 "Branching and Subscreens") when R$R is set to '++' or '%%'. When R$R is set to '--', TRANS replaces its current branch history with the stored copy before performing its return branch. This results in a return branch to the bookmarked screen

Setting R$R = '++' will save the current branch history list. Setting R$R = '%%' will save the current history list and add the current screen to it. Setting R$R = '--' will restore the saved history (replacing and erasing the history of the most recent branches) and branch back to the last entry on that restored list, which is the screen you were on before the screen where R$R was set to '++', or the screen you were on when R$R was set to '%%'.

This feature provides an easy way to return to a specific screen, irrespective of the number of branches made after leaving that screen. In situations where many screens can branch to a common screen (for example, a help screen or some other utility that is common throughout an application).

16.3 Cursor Control: C$C and C$MULREC

As described in Section 5.5.2 “Editable”, the order of the editable fields in the field names section of the screen description (TRS) provides TRANS with the order of the cursor stops on the screen. The need may arise for altering this order of cursor stops contingent on the actual data being entered. A facility for controlling the cursor movement is provided via the special local RMO field "C$C". This field, an alphanumeric field that can be up to A18 in length,can be set to the name of an editable field on the screen and TRANS will move the cursor to the named field rather than move the cursor to the next field from the field names part of the TRS.

In multi-record screens, you can specify which record in the repeating portion of the screen the cursor should go to by setting its number in the special local integer RMO field “C$MULREC”.

If C$MULREC is set to “4”, for example, the cursor will be placed at the fourth record in the multirecord screen, in the field identified by the C$C field.

If a Check statement evaluates to true, i.e. finds an error, after the RMO has set C$C to a field name, the cursor remains at the field last entered and does not go to the field referenced by C$C after the error is cleared. However, by pressing ENTER after clearing the error, the user can move the cursor to the C$C field referenced.

16.3.1 Example of Cursor Control

We have the following tax payment record.

```
*        TAXPAY.DEF
MAS 10000
ACCT# X99999 KEY1    "account number"
OWED84 D2            "1984 taxes owed"
PAID84 D2            "1984 taxes paid"
OWED85 D2            "1985 taxes owed"
PAID85 D2            "1985 taxes paid"
```

The operator posting payments will enter an account number. If that account has zero in PAID84 but a non-zero value in OWED84 the cursor will go to the PAID84 field for payment entry. Otherwise, if PAID84 is non-zero, the cursor will go to PAID85 for payment entry. If the payment entered into PAID84 exceeds the amount owed for that year, the difference will be applied to PAID85.

```
*       TAXPAY.TRS
*
TAX TAXPAY.MAS 1 TAXPAY.RMO MATCH
E ACCT#
D OWED84
ER PAID84/D2
D OWED85
ER PAID85/D2
SCREEN
CE ENTER TAX PAYMENTS
BL
ACCOUNT NUMBER: ACCT-
BL
-------- 84 --------     -------- 85 --------
OWED:  -------OWED84    OWED:  -------OWED85
PAID:  -------PAID84    PAID:  -------PAID85
END

*      TAXPAY.RMS
*
FILE TAXPAY.MAS
LOCAL
S$S/A6
M$M/A2
C$C/A6
PROGRAM
IF M$M NE 'UP' THEN GOTO DONE END
IF S$S NE 'BEGREC' THEN GOTO NOTBEG END
*
* At BEGREC, set the cursor to PAID84 or
* PAID85 depending on OWED84.
*
IF OWED84 NE 0 AND PAID84 EQ 0 THEN
  C$C = 'PAID84' ELSE C$C = 'PAID85' END
*
NOTBEG: IF S$S NE 'PAID84' THEN GOTO DONE END
*
* If the payment entered in PAID84 is greater
* then the amount owed for 84, apply the
* difference to 85.
*
IF PAID84 GT OWED84 THEN PAID85 = PAID84 - OWED84 ;
  PAID84 = OWED84 END
DONE: STOP
```

16.4 Controlling the Skipping of Fields: SK$SK

TRANS normally issues calls to the RMO at "BEGREC", "EOFREC", or when the user has entered data into a field. TRANS can also be instructed to issue an RMO call any time the user tries to skip past a field, i.e. by pressing ENTER, carriage return, or one of the directional arrows. This feature provides TRANS with a mechanism which blocks the user from skipping past required data entry fields, and gives additional information to the RMO about the location of the cursor. SK$SK may be useful when the REQUIRE statement (see Section 5.5.5 “REQUIRE Statement”) is not appropriate.

To use this feature, the RMO must have a local integer field SK$SK. If SK$SK is in the RMO, then trying to "skip" past a field causes TRANS to call the RMO with the following values automatically set:

1. S$S is set to the name of the field being skipped
2. SK$SK is set to "1"
3. M$M is set to the pre-link call, e.g. "UX", "IX", or "AX".

During all other (non-field skipping) RMO calls, SK$SK is set to "0" (zero). The RMO should **never** set SK$SK to zero or 1.

When SK$SK is in the RMO and the user skips past a field, links are only processed if the RMO changes the value of the field. If the RMO leaves the field intact, links are **not** processed, nor is the post-link RMO call given. (Because the post-link RMO call may not be issued after a field-skip call, care should be taken to ensure that pre-link field-skipping RMO logic does not set any Check statements, which are only tried after a post-link RMO call.)

At the SK$SK call the RMO can change the field being skipped, use C$C to move the cursor to another field, issue an automatic branch, or perform most of its usual functions. The field log is updated if the RMO changes a loggable field that is being skipped.

When the RMO sets SK$SK to "2", this indicates to TRANS that the field may be skipped, i.e. that TRANS should process the field skipping keystroke in the normal manner. The RMO should only set SK$SK equal to "2" if the field may be skipped **and** if the current value of SK$SK is "1".

The skipping logic has some consequences which may not be evident.

First, if the RMO behind the screen includes SK$SK as a local field, then the RMO **must** manage SK$SK.

Second, if the RMO sets the field being skipped to the same value it contained when the RMO was called, i.e. does not **change** the field's value, then SK$SK must be set to 2 in order to be able to skip the field. This is because TRANS does not know the RMO set the field; it only knows whether the field has been changed (either by the user or the RMO). When the RMO sets the field equal to its previous value, then, it is the same as if the user had skipped the field and the RMO had done nothing. Therefore, SK$SK must be set to 2, to allow skipping the field.

Third, if the RMO changes the value of the field when the user skips it, then the RMO should **not** set SK$SK to 2. If SK$SK is erroneously set to 2, the value set by the RMO will be updated in the file, but the new value of the field will not be refreshed on the screen. This is because when SK$SK equals 2, TRANS assumes that the field was skipped, and does not refresh the field on the screen.

Fourth, SK$SK may not be used with multi-record screens.

16.4.1 SK$SK Example

The following example uses SK$SK to require entry into the field EMP#.

```
FILE PAYROLL.MAS
LOCAL
S$S/A6
M$M/A2
SK$SK/I
...
PROGRAM
*
IF M$M NE 'UX' THEN GOTO POSTLINK END
*
* All other fields except EMP# may be skipped.
*
IF S$S NE 'EMP#' AND SK$SK EQ 1 THEN SK$SK = 2 ;
 GOTO DONE END
...
POSTLINK: ...
...
DONE: STOP
```

To summarize, in an RMO containing the local integer field SK$SK:

1. When data is entered in a field the RMO receives the usual calls with SK$SK set to zero.
2. If the user presses ENTER, carriage return, or a directional arrow to skip past a field, then: TRANS automatically sets SK$SK to "1", S$S to the name of the field being skipped, M$M to the pre-link mode, and then the RMO is called. The cursor will not move until:
   a. The RMO sets SK$SK to "2", or
   b. The RMO changes the value of the field, or
   c. The RMO sets C$C.

These precautions should be understood before implementing the SK$SK functionality.

1. If SK$SK/I is in the RMO, then the RMO **must** set SK$SK to "2" at all skippable fields if the RMO is called with SK$SK set to "1" and S$S set to a skippable field.
2. Do **NOT** set SK$SK to anything other than "2".
3. Do **NOT** set SK$SK to "2" **unless SK$SK was "1".**
4. To allow skipping a field, the RMO **must** either set SK$SK to "2", or **change** the value of the field.

16.5 Highlighting Fields

Data fields can be highlighted by using the display attributes for bold, underline, blink and reverse video, or any combination of these.

Highlighting is effected from two local arrays in the RMO behind the screen, "H$NAME/An(n)"[3] and "H$CODE/I(n)". Before displaying any field TRANS scans the H$NAME local array for a field name match. If found, the corresponding H$CODE integer value is used. This value, between 1 and 15, is taken as the sum of four settings.

```
1 bold     2 underline     4 blink     8 reverse video
```

For example, to highlight a field using bold and blink, the H$CODE value is "5".

The data field is then highlighted accordingly. The search of H$NAME local array stops when a corresponding H$CODE element of zero is found. Setting H$CODE to "-1" will return the field to the normal highlighting prescribed by the setting in the logical name OPTION (see Section 5.10 "Video Highlighting Facilities").

AdmTrans for Windows maps these traditional character-cell highlighting attributes to specific color schemes by default, but these default mappings can be overridden in the TRANS environment file, as described in Section 6.17.13.7 ""Video" Highlighting in TRANS for Windows". For example, by default the video attribute "bold" is rendered as darkred on a white background.

H$CODE can also be set to one of the user-defined color schemes set in the TRANS Environment File (via *field.color_NN.background* and *field.color_NN.foreground*) as described in Section 6.17.13.4 "FIELD keywords". To use the user defined codes in H$CODE, set H$CODE to 1000 + NN, or 2000 + NN if you want to reverse background and foreground.

16.5.1 Highlighting Example

The following example illustrates the highlighting of the NETDUE field in a customer receivable inquiry screen. The NETDUE field is always displayed in user-defined color scheme 46, however, if there is a balance due, the NETDUE is displayed in color scheme 47.

```
FILE CUSTOMER.MAS
LOCAL
H$NAME/A8(2) 'NETDUE'
H$CODE/I(2) 0 0
PROGRAM
IF NETDUE EQ 0 THEN H$CODE(1) = 1046 ELSE H$CODE(1) = 1047 END
```

3. H$NAME may be specified with a field type up to size A18.

16.6 Printing Messages: P$P

The RMO behind a screen can be used to print messages on a hard copy printing device. Whenever the RMO sets P$P, an alphanumeric local field, to an alphanumeric string, the contents of P$P are printed immediately on the device assigned to the logical name ADM$PRTn. TTn or SPn, placed on the screen header line of the screen description, as described in Section 5.3.1.9 “SPn or TTn: Print Device Specification”, determines the value of "n".

If the RMO sets P$P, then after the contents of P$P are printed, **the RMO is called again** and additional print lines can be placed in P$P. The printing continues until the RMO does not set P$P. In this way messages containing several lines can be printed on a hard copy printing device.

TRANS can be instructed to de-allocate[4] a device that has been allocated by TRANS for printing the contents of P$P, by setting P$P to "&&", as follows:

```
IF LINE(J) EQ ' ' THEN P$P = '&&' END
```

When P$P is set to "&&" TRANS will de-allocate the channel to the device to which P$P has been printing, and the RMO **will not be called again.** If TRANS is not instructed to deallocate the device in this manner the device will remain allocated until TRANS is exited.

The first character of P$P serves as a carriage control character for the vertical spacing on the hard copy printing device. The carriage control characters are:

```
"blank" causes a vertical space before printing
"0" causes a vertical double space before printing
"1" causes a skip to top of form before printing
"+" causes no vertical space before printing
```

The "^" character, which is converted to blank on output in ADMINS,[5] can be used in constants to create lead blanks in the P$P string. Also the "^" character can be used with the NCAT subroutine (see Appendix H.3.1 “NCAT - Concatenating fields” for a description of concatenating fields using NCAT) when concatenating fields with multiple blanks in P$P. (Single character blank fields are not removed by NCAT.)

16.6.1 Example Printing a Tax Bill Validation

A screen is being used to enter tax payments. After each payment entry a message is to be printed on TTn containing the account number, the date and the amount paid. (TTn could be a passbook/document printer into which a tax bill is inserted.) The screen header line references PAYM.RMO as the active record maintenance procedure, and the screen header line also contains the "TTn" keyword, e.g. TT4.

```
*   PAYM.RMS
*
FILE PAYM.MAS
LOCAL
S$S/A6
M$M/A2
P$P/A30
TODAY/DA
BLANK/A1
PROGRAM
```

4. De-allocating a device makes it available to other users.
5. see Section 2.4.2 “Field Data Types”

```
IF S$S NE 'AMT' OR M$M NE 'UP' THEN GOTO DONE END
P$P = NCAT(P$P,BLANK,ACCT,BLANK,TODAY,BLANK,AMT)
DONE: STOP
```

The above RMS would print the account, date and payment amount immediately after a value was entered into the AMT field.

16.7 Top of File Control: F$F

An RMO behind a screen can use the F$F field to instruct TRANS at the "BEGREC" call to go to top of file rather than display the current record. The first record in the file is displayed instead. F$F is an local integer field which, when set to "1", causes TRANS to display the top of file record. F$F is immediately reset to zero by TRANS whenever the RMO sets it to one.

16.7.1 Example Using F$F To Secure Student Records

The registrar wishes to place a terminal in a public place to allow students to examine their grades. The registrar wishes to allow a student to examine only his/her own grades, and not have access to the grades of other students. The grade file looks as follows:

```
*      GRADES.DEF
*
MAS 4000
SS# X999999999 KEY1       "Students Social Security Number
BIRTHDAY DA KEY2          "Students birthday, the password"
SNAME A30                 "students name"
COURSE1 A10               "first course"
GRADE1 A1                 "grade in first course"
COURSE2 A10               "second course"
GRADE2 A1                 "grade in second course"
...
COURSE10 A10              "tenth course"
GRADE10 A1                "grade in tenth course"
```

The registrar sets up a file keyed on social security number and students birthday. The birthday serves as a password, i.e. it is assumed that the student alone knows his/her own birthday.

The following screen, which uses the MATCH (see Section 5.3.1.8 "MATCH: Require Exact Match") keyword and the accompanying RMO which uses the F$F field, will allow a student to examine only his/her own grades by entering their social security number and birthday (password).

Mis-matching on the key, (e.g. by entering an incorrect birthday), or using the NEXT, NREC, PREV or other keystrokes to browse through the file, will all send the screen to the top of file where a null record will be displayed.

```
*      GRADES.TRS
GRADES GRADES.MAS 1 GRADES.RMO NOMSG MATCH
E SS#
E BIRTHDAY
D COURSE1
D GRADE1
D COURSE2
D GRADE2
...
D COURSE10
D GRADE10
SCREEN
ENTER YOUR SOCIAL SECURITY NUMBER AND YOUR BIRTHDAY
EXAMINE YOUR GRADES, AND THEN PRESS 'NEXT' TO
CLEAR THE SCREEN.
BL
SOCIAL SECURITY#: SS#------      BIRTHDAY-----
BL
COURSE1---  GRADE1-
COURSE2---  GRADE2-
...
COURSE10--  GRADE10-
END
*      GRADES.RMS
*
FILE GRADES.MAS
LOCAL
S$S/A6
M$M/A2
F$F/I
LAST/A6
PROGRAM
IF S$S EQ 'BEGREC' THEN GOTO BEG END
IF S$S EQ 'EOFREC' THEN GOTO EOF END
LAST = S$S
EOF: STOP
*
* ONLY DISPLAY THE RECORD IF LAST FIELD ENTERED WAS BIRTHDAY
*
BEG: IF LAST NE 'BIRTHD' THEN F$F = 1 END
LAST = ' '
```

16.8 Post-Writeback EOFREC RMO Call: B$OB

During normal end of record processing, the RMO behind the screen is only called once, and that call precedes writing to any disk files. However, the application procedure may need confirmation that the writing to the disk files was successful. This is done by requesting a second EOFREC call.[6]

If during the normal EOFREC RMO call, the RMO sets a local integer field B$OB to "1", then TRANS will call the RMO again after doing all the end of record writing to all the files. At this point, the application can record that all files were updated successfully.

The status (S$S) of the second call is "EOFREC" and the mode (M$M) is the same as the normal EOFREC call, e.g. "UP". Care must be taken to set B$OB to "1" during the first EOFREC call only, and the RMO logic must determine when it is processing the second EOFREC call.

16.8.1 Using B$OB

The RMO behind the screen should contain the following logic to use the second EOFREC call:

```
FILE ...
LOCAL
M$M/A2
S$S/A6
B$OB/I
SW/I
...
PROGRAM
IF S$S EQ 'EOFREC' AND SW EQ 0 THEN GOTO FIRST END
IF S$S EQ 'EOFREC' AND SW EQ 1 THEN GOTO SECOND END
...
GOTO OUT
*
FIRST: B$OB = 1 ; SW = 1 ;
...
GOTO OUT
*
SECOND: SW = 0 ;
...
OUT: STOP
```

6. If B$OB is set to obtain a second EOFREC call, the two EOFREC calls can be used as "pre-link" and "post-link" calls even though, M$M is set to a post-link status at both EOFREC calls. The RMO can change the LINK key fields during the first EOFREC call (when B$OB is set). Before the second EOFREC call, TRANS will re-try the LINKs. At the second EOFREC call, the RMO can set new values for LINK fields which will be written back to the disk after that call.

16.9 Look Ahead: NX$fieldname

One can examine the values of the record following the last record currently displayed on the screen in TRANS. For each field that is to be examined create a local field called "NX$fieldname". For example, to examine ACCT create NX$ACCT.

The "NX$fieldname" field will always be set to the value of "fieldname" from the record following the last record currently displayed on the screen. If the last record on the screen is also the last record in the file then an integer field called "NX$EOF" will be set to "-1" and the "NX$fieldname" values will remain unchanged from their last setting.

16.9.1 Look Ahead Example

Look ahead is particularly useful in a multi-record screen (see Section 16.22 "Multi-Record RMO Support" for an additional discussion of multi-record RMO support) with a BREAK where all of the records for a specific BREAK field may sometimes not fit on a single screen. If the operator was in the habit of only looking at a single screen, some of the detail for a key might be missed. The "NX$fieldname" could be used to give the operator a message that more records for the same key exist.

The following example checks the next record, and if it belongs to the same account, places a message on the screen indicating there are additional detail records for the same account.

```
Name: NAME------------------------  Net Due: NETDUE--------
BL
     MORE-----------------------------------------------
BL
 Account     Date     ChgCode    Amount     Paid    Net Due
---------  ---------  --------  --------  --------  --------
ACCT-----  -------RD  CCD-----  ------OA  ------PD  ------ND
END     *   DETAIL.TRS
*
DETAIL DETAIL.MAS 16 DETAIL.RMO NOMSG BREAK ACCT
*
LINK CUSTOMER.MAS
K ACCT
L NAME
L NETDUE
END
*
D NAME
D NETDUE
DR MORE/A50
E ACCT
D RD
D CCD
D OA
D PD
D ND
*
SCREEN
                    BILLING DETAILS
BL


*  DETAIL.RMS
*
FILE DETAIL.MAS
LOCAL
M$M/A2
S$S/A6
```

```
NX$ACCT/X999999999
NX$EOF/I
MORE/A50
MSG/A50 '* There are more detail records for this account *'
PROGRAM
MORE = ' '
IF S$S NE 'PGBRK' OR NX$EOF EQ -1 THEN GOTO DONE END
IF NX$ACCT EQ ACCT THEN MORE = MSG END
DONE: STOP
```

16.10 Select Records: S$SEL

A facility whereby TRANS can apply selection criteria to records in a file in order to choose which records to display adds greatly to TRANS' utility as an inquiry tool.

TRANS can already retrieve records via their key value. Record selection enhances TRANS further as a more general retrieval tool.

First, we will describe the implementation of a record selection facility.

Then we will offer some cautionary words about the use of this facility because of its potential negative effect on overall system throughput. The need for caution arises out of the potential for the user of a record selection screen requesting large amounts of sequential file searching at the mere press of a keystroke.

Finally, we will present an example of the application of record selection.

16.10.1 Implementation of Record Selection

The implementation technique chosen for record selection uses a local integer field S$SEL in an RMO operating with the screen to instruct TRANS to display or bypass the record just read. The "S$SEL" field, an integer, is set to "1" at "BEGREC" to select (display) the record, and is set to "0" (zero) to pass over the record. In the latter case, when a record is passed over, i.e. not displayed, the next sequential record is read and another "BEGREC" call is issued to the RMO. If the whole file is read without finding any record to display then the first record in the file is displayed. When the local field W$W is used in an RMO with record selection, setting W$W will write back to disk, even if S$SEL is set to "0".

The RMO can also set S$SEL to a "2" to instruct TRANS **not** to display the record **and** treat this record as the last record on a page of a multi-record screen. That is, to simulate a page break.

Alternatively, the RMO can set S$SEL to "3" to instruct TRANS to display the record and to treat the record as the last record on a page of a multi-record screen, i.e. to simulate a page break after displaying the record.

16.10.2 Caution In The Use of Record Selection

Record selection can allow the user to execute long sequential searches through files in response to a single keystroke (e.g. the NEXT keystroke), perhaps without the user even quite realizing what is happening. There are various ways to protect against misuse of record selection by the proper design of the RMO controlling the selection process. Possible techniques would include:

1. Implementing an explicit request procedure for activation of the selection mechanism. For example, have the user enter a letter into a local request field.
2. Have the RMO stop the search after n records are examined rather than proceeding to the end of file until the criteria is satisfied. The search can always be stopped by selecting the nth record, but displaying a message saying the record did **not** actually meet the search criteria. Another way to terminate the search would be to use F$F, (see Section 16.7 "Top of File Control: F$F"), to display the first record in the file.

16.10.3 Example Of Record Selection

An accountant is examining an accounts receivable file. The accountant wishes to display those records where the open balance is above a certain selection criteria value. The selection criteria value will be altered as the file is examined.

```
*     RECEIV.DEF
*
MAS 1000
CUST# X99999 KEY1  "Customer Number"
TOTALDUE D2        "Total owed"
LASTPAY D2         "Amount of last payment"
LASTDA DA          "Date of last payment"
TOTPAY D2          "Total paid this year"

*    REC.TRS
*
REC RECEIV.MAS 1 REC.RMO NOMSG
ER SEARCH/D2
E CUST#
D TOTALDUE
D LASTPAY
D LASTDA
D TOTPAY
SCREEN
CE ACCOUNTS RECEIVABLE DISPLAY SCREEN
BL
     SELECT ACCOUNTS OWING ----SEARCH OR MORE
           ZERO MEANS SHOW ALL RECORDS
BL
CUSTOMER NUMBER: CUST-
AMOUNT OWED: ------TOTALD    TOTAL PAID THIS YEAR: ----TOTPAY
BL
DATE OF LAST PAYMENT:      LASTDA---
BL
AMOUNT OF LAST PAYMENT:  ----LASTPAY
END
```

```
*      REC.RMS
*
FILE RECEIV.MAS
LOCAL
S$S/A6
M$M/A2
S$SEL/I
SEARCH/D2
PROGRAM
IF S$S NE 'BEGREC' THEN GOTO DONE END
S$SEL = 0
IF SEARCH EQ 0 OR SEARCH LE TOTALDUE THEN S$SEL = 1 END
DONE: STOP
```

16.11 Status Line Control: M$MSG and M$LOC

The RMO behind the screen can control the content and location of a "status line", using the special local RMO fields M$MSG/An (message text), and M$LOC/I (optional line number for message). Whenever the RMO changes the value of M$MSG, it is re-displayed. Use the TRANS Environment File entry:

```
m$msg.position=S
```

to have the M$MSG contents displayed on the Windows status line within the TRANS display window[7]. The status line can be used for such purposes as basic help for the user, an application title, date and time, or subtotals on a multi- record screen (for subtotals, set M$MSG at the PGBRK RMO call). For example, to put the current date and time on the status line, first declare TODAY and NOW as DR fields in the TRS, then insert the following in the RMO:

```
LOCAL
.
.
* Content and Location for Status LINE
* Value shown below places Status line at line 20
*
M$MSG/A20
M$LOC/I 20
*
TODAY/DA
NOW/A8
BL/A1 ''
.
.
PROGRAM
M$MSG = NCAT(M$MSG,TODAY,BL,NOW)
.
.
```

7. Without the TRANS Environment File entry the M$MSG content displays inthe mainTRANS window, at the line designated by the value of the M$LOC field.

16.12 Check Screen Exit Keystroke: E$NDSCR

The special field E$NDSCR/A2 can be used to check which manual (keystroke) method was used to attempt to exit the current screen. At the last RMO call before branching (the EOFREC call on the current record), E$NDSCR is set by TRANS to the manual branch code that was typed by the user, i.e. "A" if TAB was followed by an "A". If the EXIT key is typed (to attempt to exit TRANS completely), E$NDSCR will have the value "CB"; and if XRET is typed (to return to the screen previous to the current screen), E$NDSCR will have the value "RR".

The E$NDSCR facility, in combination with B$B (see Section 16.2 "Automatic Branching: B$B and R$R") can be used to enable flexible control of the user's access and movement within a family of screens. At EOFREC, the value of E$NDSCR can be checked for validity, and, if the RMO sets B$B at the EOFREC call, the manual branch, EXIT keystroke, or XRET keystroke can be overridden.

16.13 ADM$MPOS: Detecting Right Mouse Button

If both the fields ADM$MPOS/I(2) and F$UNCKEY/A4 (see Section 16.15 "F$UNCKEY - Function Key Detection in RMO") are present, the RMO will get a call when the user presses the right mouse button. F$UNCKEY is set to 'msf3'and M$M to 'FX'. If the mouse pointer is outside a field, S$S will be blank, and ADM$MPOS(1) will contain the line number and ADM$MPOS(2) the column number where the mouse pointer was when clicked.

If the mouse pointer is within a field, S$S will contain the field name, and ADM$MPOS will contain the position within the field. If ADM$MPOS is declared with dimension 4 (e.g.ADM$MPOS/I(4)), ADM$MPOS(3) will contain the line number, and ADM$MPOS(4) will contain the column number where the field starts. If ADM$MPOS(5) is present it is set to 1 if the user double-clicked in a field.

16.14 S$BL-Detect "blank" typed into numeric field

The special field S$BL/I will be set to 1 (one) if the user entered only spaces (followed by Return/Tab) into a numeric field, otherwise it will be set to 0 (zero). This makes it possible for the RMO to detect whether a space/Return or a 0/Return was entered in a numeric field (as both cases result in a value of zero being stored in the field.)

16.15 F$UNCKEY - Function Key Detection in RMO

If the RMO behind a screen has the local field F$UNCKEY (type A4) defined, TRANS will place into F$UNCKEY a unique symbolic value for the function key that terminates any entry typed into an editable field in the screen. In addition, if a function key is pressed without entering or editing a field (for example, if nothing is typed into a field and the right arrow function key is pressed to move to the next editable field) the RMO gets a special call with M$M set to 'FX' and S$S set to the name of the field at the current cursor position, and F$UNCKEY set to show which keystroke was presssed ('rght' in the example).

The values returned by TRANS into F$UNCKEY are the names of TRANS keystrokes functions if the keystroke is part of the TRANS environment, i.e. if a TRANS keystroke function has been mapped to it. If the function key is not part of the TRANS environment TRANS will put the "standard function keystroke" name into F$UNCKEY.[8]

If a selection is made in a LOOKUP window that RETURNS a value the RMO is called as if the field had been typed into, and F$UNCKEY is set to "LKUP". If HOME is used to exit a LOOKUP window for which CR_EXIT has been specified, the RMO is called with mode set to "FX" and F$UNCKEY set to "LRET" (see Section 5.11 “LOOKUP Window”).

If a text field window is exited in either of the following circumstances: 1) via "quit" (no changes to the document are saved); 2) the window was open on a display-only field; the RMO is called with mode set to "FX" and F$UNCKEY set to "TRET". If a text field is changed (if the field is editable and you leave the window via "exit") then the RMO is called as usual when a field has been typed into.

The TRANS functions "prt" (print screen) and "ref" (refresh screen) are NOT "trapped" as "function keys" when they are used by themselves. (The RMO is NEVER called with M$M set to "FX" and F$UNCKEY set to "prt" or "ref".) However, these two keystrokes are "trapped" as terminator keystrokes (i.e. when S$S is set to the name of the field just typed into. **Provided only a function key was pressed** (i.e. M$M EQ 'FX') you may, in the RMO, instruct TRANS to ignore the function keystroke, or to reinterpret the function keystroke, by resetting F$UNCKEY to one of the following values:

Value	Action
SKIP	Tell TRANS to ignore the keystroke.
RET	Tell TRANS to act as if <C.R> was pressed.

8. This behavior is changed if the statement "f$unckey=physical" is present in the TRANS environment file. "f$unckey=physical" tells TRANS to always load the standard function keystroke name into F$UNCKEY, whether or not the keystroke is part of the TRANS environment (see Section 6.17.5 “F$UNCKEY=PHYSICAL, Load F$UNCKEY with Physical Key Names”). The TRANS standard keystrokes are described in Section 6.2 “Standard Functional Keystrokes”. The TRANS environment is described in Section 6.17 “The TRANS Environment File”. The names of TRANS standard keystrokes are always returned to F$UNCKEY as all lowercase, e.g. "exit", "home", "menu". Standard function KEY_NAMEs are always returned to F$UNCKEY as all uppercase, e.g. "CT_B", "HOME", "F16".

Value	Action
HELP	Invoke TRANS HELP (Act as if HELP was pressed).
REFR	Refresh the DR and ER fields.
DONE	Ignore ***lookup.rightclick*** behavior at ADM$MPOS-induced RMO calls.(see Section 6.17.13.1 "TRANS main program")

If C$C is set when M$M = 'FX' then TRANS acts as if F$UNCKEY was set to 'SKIP'. TRANS ignores the keystroke, and the cursor is placed at the field indicated by C$C.

If at least one regular character was typed into the field on the screen before a function key is pressed, the RMO is called normally (i.e. S$S is set to the field's name and M$M is set to the current mode). F$UNCKEY is still set with the appropriate symbolic value, but TRANS acts as if the input string was terminated by a RETURN (Carriage Return). You cannot reset F$UNCKEY in this case, to cause the function key to be ignored or reinterpreted, but since you can, in the RMO, detect which function key was pressed (TRANS puts its symbol into F$UNCKEY), you may take any action required, e.g. if an input string is terminated by an UP arrow instead of a RETURN, you might use C$C to move the cursor to the previous, instead of the next editable field.

16.15.1 F$UNCKEY - Example

The following RMS causes the current record to be deleted if the Remove keystroke is pressed when the cursor is at the key field (EMPL#):

```
FILE ADM$DEMO:PERSONNEL.MAS
*
M$M/A2
S$S/A6
F$UNCKEY/A4
STAT/I
CTRLD/I(3)  4 4 0
*


PROGRAM
*
*  Delete current record, if cursor at key field <remove>
*  is pressed.
*
IF ((M$M EQ 'FX') AND (S$S EQ 'EMPL#') AND (F$UNCKEY EQ 'REMO'))
    THEN STAT = SETKEY(CTRLD) END
```

16.16 Subscreen Status and Control: ADM$SUBSCR

The special field ADM$SUBSCR/A18 provides the RMO with subscreen status and control.[9] If ADM$SUBSCR is declared as an A18 field, TRANS sets it to the current subscreen name or "MAIN", if there is no current subscreen. To change to another

9. See Section 5.15 "Subscreens".

subscreen, the RMO may set ADM$SUBSCR to the new subscreen name. Note that C$C[10] can be set in the same RMO call as ADM$SUBSCR, to change subscreens and put the cursor at some field other than the first editable field in the subscreen.

Since the main screen will typically have very little on it when subscreens are in use, it will often be desirable for one of the subscreens to appear immediately when the user enters the screen. This is accomplished by setting ADM$SUBSCR at the post-link BEGREC RMO call.

If ADM$SUBSCR is present, then whenever TRANS switches from one subscreen to another, or between the main screen and a subscreen, there is a special RMO call. S$S is set to "BEGSCR"; M$M is set to "UP", "AP", or "IN"; and ADM$SUBSCR contains the new subscreen name. This special call can be used, for example, to set values of fields in the subscreen, to set C$C, or to call the EDFLDS subroutine, etc. This BEGSCR RMO call and the ADM$SUBSCR field can be used together to display several subscreens in succession: at the a BEGSCR call, set ADM$SUBSCR to another subscreen name to display that subscreen immediately after the current one is displayed.

16.17 ADM$ENTER: Force TRANS Field Entry Processing

The special integer field ADM$ENTER provides easy method to have TRANS act as if a field had been entered, without actually entering any field (or simulating entering a field). This facility is useful to retry links when link keys are changed, but no field has been entered (links ordinarily are only retried if a link's KC or C field is typed into), to refresh fields on the screen, or to cause additional RMO calls.

Setting ADM$ENTER to 1 in the RMO makes TRANS behave as though an editable field on the screen named ADM$ENTER had been typed (and changed) by the user.

Normally, ADM$ENTER is set at a post-link RMO call. All normal processing occurs after the post-link call. Then, at the point where TRANS would normally wait for user input at the next field, TRANS instead sets ADM$ENTER to zero and issues pre-link and post-link RMO calls with S$S set to 'ADM$ENTER' (truncated to 'ADM$EN' if S$S is A6). After the ADM$ENTER calls, all normal processing occurs, just as if a field had been entered. (If ADM$ENTER is set at a pre-link call, the normal post-link call occurs, followed by the pair of S$S = ADM$ENTER calls.)

The RMO can force LINKs to be retried by making ADM$ENTER a C field in one or more LINK paragraphs.[11] LINK KC fields can be set at a pre-link ADM$ENTER call. The RMO can get another pair of calls by setting ADM$ENTER again at an ADM$ENTER call. C$C[12] (cursor control) can be set either before or at ADM$ENTER calls: it takes effect after the ADM$ENTER calls occur.

10. See Section 16.3 “Cursor Control: C$C and C$MULREC”
11. See Section 5.4 “External Files”
12. See Section 16.3 “Cursor Control: C$C and C$MULREC”

16.18 Special Keystroke to Call the RMO

TRANS can be instructed to perform a special RMO call, regardless of where the cursor is, by using the RMO[13] keystroke (either manually or via SETKEY). This feature is enabled if the character "7" is included in the string assigned to the logical name OPTION (see Appendix A: "Options").

If enabled, and the RMO key is pressed before entry into a field, TRANS calls the RMO with S$S set to the field the cursor is on, and M$M set to 'XX'. This feature can be used, for example, to trigger special RMO processing regardless of where the cursor is, or processing that would not occur at any of the other RMO calls.

16.19 Using the RMO with Table Driven Error Messages

Error messages for Check statements can be placed in an ADMINS data file keyed on an integer error message code number, with an alphanumeric field containing the message text. Using table driven error messages involves several screen components:

1. An error message table (see Section 5.5.6.1 "Table Driven Check Statement Error Messages");
2. A LINK paragraph in the TRS (see Section 5.4.1 "LINK Paragraph");
3. Check statement(s) in the TRS (see Section 5.5.6.1 "Table Driven Check Statement Error Messages" and Section 16.19.1 "Check Statement Syntax for Table Driven Messages");
4. Optional RMO logic "behind" the screen (see Section 16.19.2 "Error Message Table Example").

13. By default, the RMO keystroke is CTRL/L.

16.19.1 Check Statement Syntax for Table Driven Messages

Check statements can activate the link to the error message table in two ways.

The RMO "behind" the screen can set the local integer field E$RR equal to the appropriate error code number when the RMO logic detects an error condition. The TRS contains a check statement which handles errors triggered via E$RR:

```
C E$RR NE 0
Error: no message
```

E$RR may also be used in a CLF Check Statement.

When the RMO sets E$RR to a positive non-zero value, this Check statement is triggered. The link to the error message table executes using the value of E$RR set by the RMO, and the message from the table is displayed on the error message line of the screen. This is a special link that is performed after an error condition is triggered by a Check statement. If there is no record in the error message table with the code which is in E$RR, or if the E$RRMSG field in the table is blank, then the message in the TRS (above, "Error: no message") is displayed on the terminal. The E$RR field is automatically reset to zero by TRANS before the next RMO call.

The other method for triggering table driven error messages does not require an RMO and is described in Section 5.5.6.1 "Table Driven Check Statement Error Messages".

These two kinds of table driven check statements, as well as standard Check statements, may be mixed within a screen.

16.19.2 Error Message Table Example

ERRORS.TAB holds error messages used by many different screens. In particular, message 21, "Invalid Program Code for this Cost Center" and message 32, "Invalid Object Code for this Program Code" have many applications.

```
*    ERRORS.DEF
*    Table driven error message file
*
TAB 250
ERROR# I KEY1      "Error Number"
MESSAGE A60        "Error Message"
```

The contents of ERRORS.TAB are displayed below.

Error #	Message
1	Invalid Fund Code
2	Invalid Cost Center Code
3	Invalid Program Code
4	Invalid Object Code
5	Invalid Appropriation Code
...	
20	Invalid Cost Center for this Fund Code
21	Invalid Program Code for this Cost Center

Error #	Message
...	
32	Invalid Object Code for this Program Code
...	

The following budget entry screen, links to the error message table ERRORS.TAB to obtain the messages for error conditions detected in the BUDENTRY.RMO.

```
*    BUDENTRY.TRS
*
BUDENTRY BUDGET.MAS 1 BUDENTRY.RMO INSERT NOMSG
*
LINK ERRORS.TAB
KC E$RR
L MESSAGE E$RRMSG
END
*
LINK PROGRAM.TAB
KC PROG
L DESC
END
*
LINK OBJECT.TAB
KC OBJ
L DESC ODESC
END
*
E COSTCTR
E PROG
E OBJ
E BUDGET
*
DR E$RR/I
DR E$RRMSG/A60
*
C E$RR NE 0
Error: No Message
...
```

When the local field E$RR is set to a non-zero value in the RMO, then the link to ERRORS.TAB displays an error message at the bottom of the screen. When inserting a new record, BUDENTRY.RMO sets E$RR to a non-zero value when a program code other than 401 or 402 or 403 is entered for cost center code B, or when an object code greater than 300 is entered for a program code between 121 and 125.

```
* BUDENTRY.RMS
FILE BUDGET.MAS
LOCAL
M$M/A2
S$S/A6
E$RR/I
PROGRAM
...
IF M$M EQ 'IN' THEN GOTO NEWREC END
...
NEWREC: IF S$S EQ 'PROG' AND COSTCTR EQ B AND
        (PROG NE 401 AND 402 AND 403) THEN E$RR = 21 ;
        GOTO DONE END ;
        IF (S$S EQ 'OBJ') AND (PROG BET 121 AND 125)
        AND (OBJ GT 300) THEN E$RR = 32 ;
        GOTO DONE END
...
DONE: STOP
```

16.20 Calculated Branches with Variable Branch Keys

The special RMO array B$KEYFIELDS/An(n) enables a single calculated branch[14] to be used with any set of key fields, i.e. to branch to any number of screens with different key structures.

B$KEYFIELDS must be an alpha (An) array dimensioned for at least two elements (if not, then this field has no special effect).

If a calculated branch has explicit key fields in the BRANCHES paragraph, then B$KEYFIELDS has no effect and TRANS uses the explicitly stated fields to search for a record in the branch target screen. If a calculated branch does not list any key fields in the BRANCHES paragraph, TRANS will look for the B$KEYFIELDS array. If the B$KEYFIELDS array does not exist, TRANS branches with no key values, to the top of the target file. If B$KEYFIELDS does exist, its contents are used as the names of the key fields to use in the branch: e.g. the name of the first key field is in B$KEYFIELDS(1), the second is in B$KEYFIELDS(2), etc.

Field names cannot be abbreviated in the B$KEYFIELDS array. The array element after the last key field name **must be blank** (if B$KEYFIELDS(1) is blank, TRANS branches with no key value). **The developer must insure that the fields named in B$KEYFIELDS have the correct field types for the keys of the target screen.**

For example, if a TRS contains

```
BRANCHES
.
.
A B$BRANCH/XX
Branch to specified screen.
.
.
```

and the RMS contains:

```
LOCAL
.
.
B$KEYFIELDS/A10(5) 'K1' 'K2' ' '
.
.
```

then when branch A is requested, TRANS forms the branch key using the values of fields K1 and K2. The key field names in B$KEYFIELDS can be changed at any time, including the EOFREC RMO call.

14. see Section 5.7.3 “Calculated Branches”

There is a second way to use B$KEYFIELDS: instead of loading the array with key field names, it can be loaded with key field **values.** Convert the key field values **to alpha (An) format** (use NCAT or FCAT subroutine)[15] and place them in the B$KEYFIELDS array before branching.

If B$KEYFIELDS(1) is not a field name, TRANS will assume that the B$KEYFIELDS array contains key values rather than key field names. You cannot mix field names and field values in B$KEYFIELDS (TRANS will exit with a message). However, any 'value' string in B$KEYFIELDS can begin with 'A$': if it does, TRANS assumes the string is a logical name which contains a branch key value in alpha format.[16] When B$KEYFIELDS is used to contain key values rather than key names, TRANS handles the B$KEYFIELDS array the same way as it handles key values provided on the TRANS command line (see Section 6.15 “Entering TRANS On A Specific Record”).

16.21 Managing Ignored Record Locks - ADM$NOLOCK and ADM$NLREC

The special local RMO integer field ADM$NOLOCK and local RMO alphanumeric array ADM$NLREC provide information which enable the RMO running with TRANS to control the screen when record locks are ignored:[17] i.e., when the user answers "I" at the "Wait or Ignore" prompt, or when file option 'I' is invoked and the record is already locked by another user. With file option 'I', the RMO can be given complete control over record locking conflict resolution (the user isn't prompted).

16.21.1 ADM$NOLOCK: Record Lock Ignored Flag

ADM$NOLOCK/I allows the RMO to detect when the user elects to ignore a record lock at the "Wait or Ignore" prompt, or when file option 'I' is in effect and the record is already locked by another user.

If ADM$NOLOCK is declared in the RMO it is maintained automatically by TRANS, and is normally (i.e., when no locks have been ignored) set to zero. If a lock on the active file is ignored, ADM$NOLOCK is set to 1 before the BEGREC pre-link RMO call in a single record screen, or before the MULREC pre-link call in a multi-record screen. If the active record is locked successfully but the user subsequently ignores a lock on one or more LINK W files, ADM$NOLOCK is set to 1 before the post-link RMO call.

15. see Appendix H.3 “Concatenation Subroutines”
16. The only way to branch with a BLANK alpha key field value, is to assign the blank to an "A$" logical name and put the "A$" logical name, not the blank, in B$KEYFIELDS.
17. See Section 19.3 “Resolving Record Access Conflicts”

Thus, the RMO must check ADM$NOLOCK and take appropriate action at every call (or at least every post-link call) where a lock may have been ignored.

In either case, once ADM$NOLOCK is set, it remains set until the user goes to another record in the active file or branches.

16.21.2 ADM$NLREC: Identify Ignored Locks

The local alphanumeric array ADM$NLREC[18] allows the RMO to detect **which** record locks have been ignored. The RMO could then take different actions depending on which records are not locked.

If ADM$NLREC is declared in the RMO it is maintained automatically by TRANS and should not be set by the RMO.

If a record lock on the screen's main file is ignored, ADM$NLREC's first element , i.e. ADM$NLREC(1), is set to "MAIN" at the BEGREC pre-link and post-link RMO calls in a single-record screen, or at the MULREC calls in a multi-record screen.

If a record lock on a LINK W file is ignored, then at the immediately following post-link RMO call, the LINK name prefix[19] is placed in the first non-blank element of ADM$NLREC.

Several record locks may be ignored at BEGREC/MULREC or at the entry of a field which sets key or C fields for more than one LINK W. "MAIN" and/or all the LINK name prefixes for all ignored locks appear in ADM$NLREC at the post-link RMO call. A blank element in the ADM$NLREC array indicates that the array contains no more entries.

TRANS blanks out all elements of ADM$NLREC after every post-link RMO call. Therefore, ADM$NLREC must be checked at every RMO call where a record lock may have been ignored: at either the pre-link or the post-link BEGREC/MULREC call, and at post-link calls when field entry changes key fields in LINK W's.

The ADM$NLREC array should be dimensioned at least large enough for the number of LINK W's in the screen, plus two. This will enable it to contain, at maximum, "MAIN" and the link name prefix of every LINK W in the screen, plus a terminating blank element.

ADM$NLREC can have any alpha (An) field size. If "MAIN" or a LINK prefix name is too long to fit in the specified field size, it is truncated. The field size should be large enough to contain unique strings for "MAIN" and for all LINK prefix names.

---

18. ADM$NLREC can be used regardless of whether ADM$NOLOCK (see Section 16.21.1 “ADM$NOLOCK: Record Lock Ignored Flag”) is also in the RMO: these two features are independent.
19. See Section 5.4.1 “LINK Paragraph” for the "=PREFIX" link name syntax. To use ADM$NLREC without automatically renaming the link fields, use the syntax "-PREFIX" instead of "=PREFIX". The "=PREFIX" or "-PREFIX" syntax **must** be used in LINKs with writeback in order for those links to be visible to the RMO in ADM$NLREC. If for some reason the RMO does not need to know about ignored locks on a certain LINK with writeback, then that LINK need not have a prefix.

16.22 Multi-Record RMO Support

RMOs operating behind multi-record screens[20] have several uses, some example are as follows:

1. To maintain summary information in local RMO fields for display on the multi-record screen.
2. To allow record selection on multi-record screens using the S$SEL fields as described in Section 16.10.1 “Implementation of Record Selection”.
3. To allow automatic branching based on the contents of a particular record in a multi-record screen. (An automatic branch may be executed as each record is displayed.)
4. To control when LINK files are written (using W$W).

RMOs behind multi-record screens get an additional call when there is a page break on the screen. S$S is set to "PGBRK" and M$M is still set to "UP".

When the cursor is moved from record to record on a multi-record screen display, the links will be re-evaluated and the RMO will receive a pre-link and post-link "MULREC" call, i.e. S$S is set to "MULREC". The reason for re-executing the link and RMO logic is so that if a branch is taken while the cursor is at a particular record on the screen, the branch will be based on correct values.

When the user moves to a new record in a multi-record screen (when a MULREC RMO call would occur), the values of fields in that record are refreshed on the screen immediately if they have changed since the page of records was originally displayed. Fields are refreshed both in the heading part of the screen and in the repeating line(s) for the new record. This ensures that the values shown on the screen for the current record are up to date. **Any local fields displayed on the screen, or local fields which link information for display on the screen, must be calculated at MULREC as well as BEGREC.**

Local ER fields[21], which in principle **may be updated,** should **not** be used on the repeating portion of the screen. If a local or calculated field is needed for display on the repeating portion of the screen, the field should be made DR, i.e. display only.

If there is an RMO and the multi-record screen contains at least one LINK W, then there is an EOFREC call every time TRANS leaves the active record, rather than just when TRANS goes to a new page of records. At these EOFREC calls, the user may set field values and may set the special field W$W (as described in Section 16.1 “Controlling Changes Written To Disk”) to control when changes in LINKed fields are written back to the LINK file. **These EOFREC calls should not be used for any other purpose.** In particular, ER or DR fields set at EOFREC calls will generally not be refreshed on the screen.

20. see Section 5.9 “Multi-Record Screens”
21. Main file and linked-in fields that are referenced in the TRS as ER fields (because they might be changed by the RMO) may be used (and commonly are used) in the repeating portion of a multirecord screen.

16.22.1 ADM$RECNO: Record Position in Multi-Record Screen

If the special integer field ADM$RECNO is present in the RMO TRANS will set it to the current record number in a multi-record screen. ADM$RECNO is set correctly at **post-link** BEGREC and MULREC RMO calls **(not at pre-link calls).**

For example, if a record is the top record displayed on the screen, TRANS will set ADM$RECNO to 1 for that record. If a record is the seventh record displayed on the screen, TRANS will set ADM$RECNO to 7 for that record.

16.22.2 Multi-Record Summary Screens

Local RMO fields in the heading of a multi-record screen can be used to allow the RMO behind a multi-record screen to maintain summary information on the multi-record screen. That is, the heading of the multi-record screen can contain summaries (sub-totals), while the rest of the screen contains the repeating detail records.

A multi-record screen stops displaying records at a "page break". A page break occurs for any of the following reasons:

1. The requisite number of records have been displayed as per the records per screen keyword (see Section 5.3 “Screen Header Line”).
2. The BREAK (see Section 5.9.2 “BREAK In a Multi-Record Screen”) criteria have been satisfied.
3. The S$SEL local field in the RMO behind the screen causes a page break, as described in Section 16.10.1 “Implementation of Record Selection”.

TRANS will display local fields in the heading as they are altered by the RMO. Maintenance of relevant and accurate summary information in these fields is the responsibility of the RMO behind the multi-record screen.

16.22.2.1 Example of a Multi-Record Summary Screen

We have a personnel file ordered by department number containing salary per employee. We wish to create a multi-record screen showing information about one employee per line and show number of employees and total salary per screen and per department displayed thus far.

```
*      DEPT.DEF
MAS 2000
DEPT# X999 KEY1
EMPL# X99999 KEY2
NAME A30
SALARY D

*      DEPT.TRS
DEPT DEPT.MAS 5 DEPT.RMO NOMSG BREAK DEPT#
DR TEPS/I         "total employees per screen"
DR TSPS/D         "total salary per screen"
DR TEPDD/I        "total employees per department displayed"
DR TSPDD/D        "total salary per department displayed"
E DEPT#
E EMPL#
D NAME
D SALARY
SCREEN
CE DEPARTMENT REVIEW SCREEN
BL
     ---TEPS  THIS SCREEN                 -----TSPS
     --TEPDD  THIS DEPARTMENT SO FAR    ------TSPDD
```

```
BL
DEPT#  EMPL#  NAME                          SALARY
DEP-   EMP--  NAME----------------------  ------SAL
END

*      DEPT.RMS
FILE DEPT.MAS
LOCAL
S$S/A6
M$M/A2
LAST/X999
TEPS/I
TSPS/D
TEPDD/I
TSPDD/D
PROGRAM
*
* Process only at BEGREC in UP mode.
*
IF S$S NE 'BEGREC' OR M$M NE 'UP' THEN GOTO DONE END
*
* Check if this record is for the same department.
*
IF DEPT# EQ LAST THEN GOTO SAME END
*
* If it is for a new department, reset all the counters.
*
TEPDD = 1 ; TSPDD = SALARY
*
NEWPAGE: TEPS = 1 ; TSPS = SALARY
LAST = DEPT#
DONE: STOP
*
* If it is the same department, add to the counters.
*
SAME: TEPDD = TEPDD + 1 ; TSPDD = TSPDD + SALARY
TEPS = TEPS + 1 ; TSPS = TSPS + SALARY
*
* If employees per screen is greater than 5, that
*  means we are actually on a new screen.
*
IF TEPS GT 5 THEN GOTO NEWPAGE ELSE GOTO DONE END
```

The displays for department 133 which has eight employees might look as follows. (The second screen is displayed after the user presses the NEXT keystroke to the first screen.)

```
            DEPARTMENT REVIEW SCREEN

              5     THIS SCREEN                 79,600
              5     THIS DEPARTMENT SO FAR      79,600

DEP#   EMPL#    NAME                          SALARY
133    00462    JONES, DAVID                  16,300
133    05031    SMITH, PETER                  15,200
133    06428    BARNES, FRANK                 16,300
133    06513    WILSON, MARY                  17,100
133    07002    ALBERT, BOB                   14,700
```

User presses NEXT

```
            DEPARTMENT REVIEW SCREEN

              3      THIS SCREEN                49,200
              8      THIS DEPARTMENT SO FAR    128,800

DEPT#   EMPL#    NAME                          SALARY
133     07814    WORTH, SUE                    16,500
133     08007    HILL, PETER                   18,100
133     09138    JAMISON, HARRY                14,600
```

16.23 Subroutines Used with TRANS

In general, all the subroutines described in Appendix H: "Subroutines" may be used in an RMO behind the screen. The subroutines in the following list, however, have a specific functionality or special utility when used in TRANS.

Subroutine	Description	See:
ASKSCR	Prompt screen from RMO.	Appendix H.14.1 "ASKSCR: Prompt directly from RMO"
AUTOBR	Automatic branch control.	Appendix H.13.1 "AUTOBR: Automatic Branch Control"
DISPFLDS	Modify list of fields displayed.	Appendix H.13.6 "DISPFLDS: Modify Field Display List in TRANS"
EDFLDS	Modify cursor order.	Appendix H.13.7 "EDFLDS - Modify List of Editable Fields in TRANS"
EDIT	Text editing on a paragraph.	Appendix H.6.5 "EDIT: "Paragraph" Editing in TRANS"
MOVFLD	Move multiple fields between files, generalized.	Appendix H.13.15 "MOVFLD - Move Fields Among Files Accessed via TRO"
NOEK	Read with no echo.	Appendix H.13.9 "NOEK - Set TRANS to Read Next Field With No Echo"
PAUSE	Pause.	Appendix H.13.10 "PAUSE - Create a Pause in TRANS"
READBR	Make branch screens read-only.	Appendix H.13.12 "POPUP - Displaying a Popup Menu in the Screen"
SETKEY	Simulate keystrokes.	Appendix H.13.14 "SETKEY - Simulate Keystrokes in TRANS"
SPAWN	Create subprocess	Appendix H.14.14 "SPAWN - Create Subprocess from ADMINS Command"
SYNC	Synchronize processes.	Appendix H.14.17 "SYNC - Synchronize Access to a File"

Subroutine	Description	See:
TTCOM	Communicate with another terminal.	Appendix H.14.18 "TTCOM - Communication With Another Terminal"
VIEWTEXT	Display text file.	Appendix H.6.9 "VIEWTEXT: Display Text File in TRANS"

16.24 TX$INITF: Automatic Initialization of Text Fields

The general discussion of initialization of text fields that appears in Appendix J.8 "The Text Initialization File" describes the purpose and syntax of Text Initialization Files. Appendix I.4.3 "Text Fields" describes how to associate a particular text initialization file with a text field. This section describes how the RMO special local field **TX$INITF** can control which of the initialization files available in the Data Dictionary will be used to initialize a text field.

When TRANS is about to open the TED window for edit on a text field, it checks the contents of the local RMO integer field TX$INITF. If TX$INITF is present, TRANS will attempt to use its value to search the **"Initialization Files for Text Fields" Codelist Table**[22] in the ADMINS Data Dictionary. If an entry for that value is found in the codelist table TRANS will use the initialization file specified in the description field for that codelist entry to initialize the file TED is about to edit.

16.24.1 TX$INITF Example

Assume that a screen includes a text field named TEXT, and that we have written the following initialization file for it (see Appendix J.8 "The Text Initialization File"):

Customer: <%40sL$CUSTNAME> Account.: <%12sL$ACCOUNT> Date....: <%TODAY>

Further assume that we have placed an entry (code value "2") identifying this initialization file in the Data Dictionary ADM$DD_TEXT_INITFILES codelist table.

We can then have the RMO:

1. Check that we are about to edit field TEXT. If we are, check that TEXT is empty. (If it isn't empty we don't want to initialize it!)
2. Load the logical names L$ACCOUNT and L$CUSTNAME with the account number (ACCOUNT) and customer name (CUSTNAME) from the virtual record, to be automatically passed to the initialization file.
3. Use the TX$INITF field to specify initialization file number **2.**
4. The initialization file is automatically inserted at the beginning of TEXT, with the values of the L$ACCOUNT and L$CUSTOMER logical names automatically substituted, as follows:

```
Customer: Ms. Jenny Lee
Account.: X92080215588
Date....: 12-Jan-1991
```

22. This table is DD ID# CT0013, Codelist Table name is ADM$DD_TEXT_INITFILES.

The following RMO code would accomplish this:

```
.
.
TX$INITF/I
F$UNCKEY/A4
STAT/I
VALUE/A24
LOGNAM1/A24 'L$ACCOUNT'
LOGNAM2/A24 'L$CUSTNAME'
.
.
IF S$S NE 'TEXT' THEN STOP END      ! Do only for TEXT field
IF TEXT NE ' ' THEN STOP END       ! Do only if TEXT is empty
IF M$M EQ 'FX' AND F$UNCKEY EQ 'edit' THEN ;
*                                   ! If EDIT just pressed to
*                                   ! edit TEXT...
   TX$INITF = 2 ;                   ! Use initfile # 2
   VALUE = NCAT(VALUE,ACCOUNT) ;
   STAT = CRLOG(LOGNAM1,VALUE) ;    ! Create L$ACCOUNT
   STAT = CRLOG(LOGNAM2,CUSTNAME) ; ! Create L$CUSTNAME
   END
```

16.25 ADM$LRC: Log RMO Calls

If the reserved field ADM$LRC/I (for **L***og* **R***MO* **C***alls*) is present in the virtual record and set to a non-zero TRANS will log the values listed below upon entry to the RMO and exit from the RMO.

On entry to RMO:

```
M$M
S$S
F$UNCKEY
```

When the RMO exits:

```
C$C
B$B
R$R
W$W
C$MULREC
```

The values are logged in a file with the same name and directory as the RMO, with a file extension of .lrc, (i.e. if the rmo is "c:\myfiles\obj\sched_insp.rmo" the log file produced will be "c:\myfiles\obj\sched_insp.lrc"

Here's an example of the contents of a log file produced via ADM$LRC:

```
M$M S$S             F$UNCKEY   C$C             B$B R$R W$W C$MULREC
UX  ADM$LRC           RET
UP  ADM$LRC           RET
FX  N                 RET
FX  FLD               RET
FX  TEST              down
UP  EOFREC            down
UX  MULREC            down
UP  MULREC            down
FX  TEST              next
UP  EOFREC            next
UX  BEGREC            next
UP  BEGREC            next
UX  BEGREC            next
UP  BEGREC            next
```

Chapter 17:External Data Files

This section describes ADMINS commands for dealing with non-ADMINS data files. On Windows systems external files are always read from or written to disk, and then transferred to tape (if necessary) via non-ADMINS utilities.

Tools are needed to examine these files, to acquire them into ADMINS files, and to create external non-ADMINS files from data in ADMINS files. The various commands provide the following functions:

	Acquire	Create	Examine
Disk	FACQUIR TXTACQ IE	FDATAP IE	

Briefly, the function of each command is as follows:

FACQUIR	Read data from an external file into an ADMINS file.
TXTACQ	Read data from an external text file into a ADMINS file.
FDATAP	Write data to an external file from an ADMINS file.
IE	Transfer information between standard ASCII files and ADMINS data files. IE has capabilities that are especially useful for passing information to and from the data interchange formats used by many popular desktop computer applications, such as spreadsheet and graphics applications.

The remainder of this section is a complete description of the above commands.

17.1 TAP Instruction File

FACQUIR, and FDATAP require an instruction file to describe the layout of the record to be read or created in the external file. The file type of this instruction file is always ".TAP".

17.1.1 Outline of the TAP Instruction File

The general outline of a TAP instruction file is as follows. The options associated with a specific command are presented with the command.

```
BPREC  RPBLK  [NRECS]  [ASCII]     file description line
```

```
NAME  BPOS  BLEN  FORM  [OPT]     field description line(s)
```

Any line that begins with an asterisk (*) is ignored and may be used for comments. No other comment delimiters are supported.

17.1.2 TAP - File Description Line

The initial line of the TAP instruction file,

```
BPREC RPBLK [NRECS] [ASCII]
```

describes the size and characteristics of the non-ADMINS file. The first two elements, BPREC and RPBLK, must be present.

- **BPREC:** The number of bytes per external record.
- **RPBLK:** The number of records per block. If the file is not blocked, RPBLK would be 1. RPBLK is always 1 for FDATAP, i.e. files are never written "blocked" by FDATAP.

  For example, the following file description line:

  ```
  50 10
  ```

  would mean 50 bytes per logical record and 10 logical records per physical block for a physical block size of 500 bytes.

  ```
  250 40
  ```

  would mean 250 bytes per logical record and 40 logical records per physical block for a physical block size of 10,000 bytes.

  There will be cases where, rather than blocking several logical records into one physicalblock, one will find a single logical record spread across more than one physical block. For example, card image (80 byte) records, one card per physical block, with three card images per logical record. FACQUIR handles this situation correctly.[1] When logical records cross physical blocks a notation of form "1/n" is used in the RPBLK field, where n is the number of physical records used to make up one logical record. BPREC is the physical record size (block size). In our example of 3 card images per logical record, BPREC would be 80 and RPBLK would be "1/3". The file description line would read as follows:

  ```
  80 1/3
  ```

- **NRECS:** The optional specification NRECS may be used in test runs. By placing a number as the third element of the initial line, the command using the TAP will only read or write that number of records. This number is for testing purposes only and must be removed to read or write the file completely.
- **ASCII:** All commands assume the external file is EBCDIC. However ASCII files can also be read or created. This is instructed by placing the word ASCII on the first line of the TAP instruction file. For example, "160 3 ASCII" would read or write ASCII blocks of 480 bytes with 3 records per block.

1. FDATAP does not support records spread across blocks.

17.1.3 TAP - Field Description Line(s)

The field description line in the TAP instruction file relates a specified portion of the logical record in a disk file to a particular field in an ADMINS data file. Field description lines have the following syntax:

```
NAME BPOS BLEN FORM [OPT]
```

- **NAME:** The name of the field in the DEF of the ADMINS file into which this field is to be acquired or from which this field is to be written.
- **BPOS:** The starting byte position of the field in the external file record. (Regardless of whether the records are blocked or not. BPOS is relative to the beginning of the logical record.)
- **BLEN:** The length in bytes of this particular external file field.
- **FORM:** The format of this particular external file field, and is one of the following:[2]
  - **E:** The field contains EBCDIC or ASCII characters.
  - **EN:** The field contains EBCDIC or ASCII numeric digits.
  - **PD:** The field contains packed decimal data.
  - **B:**The field contains binary data.
- **OPT:** The options associated with the field description lines are different for acquiring external data and writing external data. The FACQUIR field description options are described in Section 17.2.3 “FACQUIR Field Description Options” and the DATAP/FDATAP field description options are described in Section 17.4.2 “FDATAP Field Description Options”.

17.1.4 Example Of A TAP Instruction File

```
*VENDOR.TAP - used to transfer vendor information
120 15
#VEND       2     4    EN
VENDOR     10    50    E
ADDR       80    20    E
CITYS     100    15    E
ZIP       115     5    EN
```

2. In addition to the common formats described here, FACQUIR has the capability to read other "optional" formats. These format options are described in Section 17.2.3.1 “FACQUIR Format Options”.

17.1.5 EBCDIC and ASCII Character Sets

ASCII Character Set (Hexadecimal Codes)

Char	Code	Char	Code	Char	Code	Char	Code	Char	Code
NUL	00	SUB	1A	4	34	N	4E	h	68
SOH	01	ESC	1B	5	35	O	4F	i	69
STX	02	FS	1C	6	36	P	50	j	6A
ETX	03	GS	1D	7	37	Q	51	k	6B
EOT	04	RS	1E	8	38	R	52	l	6C
ENQ	05	US	1F	9	39	S	53	m	6D
ACK	06	blank	20	:	3A	T	54	n	6E
BEL	07	!	21	;	3B	U	55	o	6F
BS	08	"	22	<	3C	V	56	p	70
HT	09	#	23	=	3D	W	57	q	71
LF	0A	$	24	>	3E	X	58	r	72
VT	0B	%	25	?	3F	Y	59	s	73
FF	0C	&	26	@	40	Z	5A	t	74
CR	0D	'	27	A	41	[	5B	u	75
SO	0E	(	28	B	42	\	5C	v	76
SI	0F	)	29	C	43	]	5D	w	77
DLE	10	*	2A	D	44	^	5E	x	78
DC1	11	+	2B	E	45	_	5F	y	79
DC2	12	,	2C	F	46	`	60	z	7A
DC3	13	-	2D	G	47	a	61	{	7B
DC4	14	.	2E	H	48	b	62	|	7C
NAK	15	/	2F	I	49	c	63	}	7D
SYN	16	0	30	J	4A	d	64	~	7E
ETB	17	1	31	K	4B	e	65	DEL	7F
CAN	18	2	32	L	4C	f	66		
EM	19	3	33	M	4D	g	67		

EBCDIC Character Set (Hexadecimal Codes)

blank	40	`	79	m	94	B	C2	S	E2
.	4B	:	7A	n	95	C	C3	T	E3
<	4C	#	7B	o	96	D	C4	U	E4
(	4D	@	7C	p	97	E	C5	V	E5
+	4E	'	7D	q	98	F	C6	W	E6
&	50	=	7E	r	99	G	C7	X	E7
!	5A	"	7F	~	A1	H	C8	Y	E8
$	5B	a	81	s	A2	I	C9	Z	E9
*	5C	b	82	t	A3	}	D0	0	F0
)	5D	c	83	u	A4	J	D1	1	F1
;	5E	d	84	v	A5	K	D2	2	F2
-	60	e	85	w	A6	L	D3	3	F3
/	61	f	86	x	A7	M	D4	4	F4
|	6A	g	87	y	A8	N	D5	5	F5
,	6B	h	88	z	A9	O	D6	6	F6
%	6C	i	89	[	AD	P	D7	7	F7
_	6D	j	91	]	BD	Q	D8	8	F8
>	6E	k	92	{	C0	R	D9	9	F9
?	6F	l	93	A	C1	\	E0		

17.2 FACQUIR: Read External File

The contents of external files can be read into an ADMINS file using FACQUIR.

17.2.1 FACQUIR: Acquire External Disk File

The FACQUIR command is used to acquire data from an external file into an ADMINS file. FACQUIR prompts for an external input file name as well as a "TAP" instruction file name describing the format of the records to be acquired. The dialogue of the FACQUIR command is as follows:

```
$ facquir
-------.TAP INPUT-FILE-NAME UPDATE RUN
TYPE:text.tap text.fil run
DATA FILE NAME:status.mas
15:32:21:42
EOF ON TEXT.FIL  50 RECORDS ACQUIRED  15:32:23:27

$
```

UPDATE	**Update** the record in the ADMINS file when the key value read from the tape record already exists. If the key balue does not already exist, **insert** the record read from tape into the ADMINS file. If UPDATE is not present, **all** records are **appended** to the ADMINS file. As with any ADMINS command that accesses records by key value, UPDATE requires that the ADMINS file be “in sort”.
RUN	The user wishes to acquire records into the data file. If RUNis not specified, then each tape record which ACQUIR reads is printed on the printer, field by field, along with the conversion of the value into its ADMINS format. By inspecting this printout the user can check out the TAP instruction file to see if every field is being acquired properly.

FACQUIR will read external disk files of any internal record format.

The record size (BPREC) used in the "TAP" file description line is the largest possible record size that FACQUIR could encounter in the file. Specifically, the external file can contain ASCII data or EBCDIC data records, blocked or unblocked. FACQUIR is often used on files received via communications lines from another computer.

FACQUIR can also be used to acquire a file created by a text editor.

17.2.2 External File Description Options

Use the following options to provide a description of the external file.

17.2.2.1 Records Spread Across Blocks

FACQUIR can handle the situation where, although logical records are blocked into physical blocks, a particular logical record positioned at the end of a physical block can be spread across a physical block boundary. ACQUIR/FACQUIR are instructed of this "spreading" condition via the use of an "S" prefix on the records per block specification (RPBLK). For example:

```
160 S512
```

The logical record size is 160 bytes, the physical block is 512 bytes, and the logical record crosses the physical boundaries. The RPBLK specification contains the physical block size prefixed with an "S" rather than the blocking factor.

17.2.2.2 Excess Bytes

FACQUIR can handle the situation where there are excess bytes at the end of a physical block that are to be ignored when there are insufficient bytes remaining to form another logical record. This is instructed by prefixing the records per block (RPBLK) specification on the TAP file description line with the letter "F" for "fill". For example:

```
120 F512
```

This means there are 120 bytes per logical record and a physical block size is 512 bytes. When the "F" prefix is present the RPBLK specification contains the actual bytes per physical block rather than the number of logical records per physical block.

```
...
```

17.2.3 FACQUIR Field Description Options

The following options are associated with the field descriptions. Multiple options may be specified on the same field description line.

17.2.3.1 FACQUIR Format Options

The following optional field format may be used on the field description line with FACQUIR:

- **ENn**: The external field contains EBCDIC or ASCII numeric digits with an imaginary decimal point.[3] "N" zeroes are appended to the external value before it is stored in the ADMINS data file. The decimal point is then placed in the resulting value according to the number of decimal places the field is defined for. Thus if the value "125" is read into a D2 field with the EN0 optional format (no zeroes added) it is stored as "1.25" (two decimal places).. If "125" is read into a D1 field with the EN2 optional format (two zeroes added) it is stored as "1250.0" (one decimal place).

  ENn should not be used when an explicit decimal point could be encountered in the external file field.

If FACQUIR is asked to read an alphanumeric field from tape, any leading blanks will be "squeezed out" in the character data. This is because ADMINS does not keep leading blanks in An fields. (If leading blanks are entered via TRANS, they are squeezed out when the data field is displayed immediately after entry.) As discussed in Section 2.4.2 "Field Data Types", ADMINS allows the insertion of the "^" character to signify leading blanks. The "^" character is displayed as a leading blank on all output representations of the alphanumeric data.

If the TAP file contains the letter "B" as an option on the field specification line then leading blanks on the tape will be converted to "^" characters as the field is being acquired. For example:

```
TITLE 30 40 E B
```

17.2.3.2 Override Byte Address Outside Record

FACQUIR always check that the TAP file doesn't reference a byte that is outside the logical record. For example:

```
120 3
...
ADDR 115 20 E
...
```

The TAP file says that ADDR starts at byte position 115 and extends for 20 bytes. However, the first line of the TAP instruction file says that there are only 120 bytes per logical record.

The letter "O" as an option on the field description line instructs FACQUIR **not** to check the byte address against the logical record size. The above example would be correct if we place the "O" on the ADDR line.

```
ADDR 115 20 E O
```

3. The optional ENn syntax is only needed if you are acquiring data from a field in the external file that has an assumed (or "imaginary") decimal point between two of its digits. If the decimal point is assumed at the extreme right of the field, or if the decimal point is explicitly present in the external file, EN (see Section 17.1.3 "TAP - Field Description Line(s)") will correctly acquire the data. The EN (by itself) syntax indicates that an explicit decimal point will be found in the external file field. If one is present it is used when the data is written to the ADMINS file. If none is found it is assumed to be at the extreme right of the field. For example, if the value "77" is read (using EN) into a field with type D2 in the ADMINS data file, the value "77.00" is stored. If the value "7.7" is read (using EN) into a D2 field "7.70" is stored.

17.2.4 TAP - SELECT Line

FACQUIR can be instructed to acquire only records that satisfy a particular selection criteria. The selection is placed anywhere in the TAP instruction file after the initial line. Only one selection may be made. There are three possible formats for the selection instruction.

```
SELECT BPOS X

SELECT BPOS AB

SELECT BPOS ABCD
```

BPOS: In all three formats, BPOS is a byte position in the tape or external disk file record.

X: The first format selects records that contain the character X in the byte position specified in the external file.

AB: The second format selects records that contain AB as the **two hexadecimal** digits starting at the specified byte position in the external file. The two hexadecimal digits comprise the code for a single EBCDIC (or ASCII) character. This format can be used to select for “non-printing” characters in the external file.

ABCD: The third format selects records that contain ABCD as the four hexadecimal digits starting at the specified byte position. These four hexadecimal digits comprise the code for two successive EBCDIC (or ASCII) characters in the external file. For example, to select For example, to select only those records in the (ASCII) external file that contain the character "A" in column 1 and the character "9" in column 2 use the following SELECT line:

```
SELECT 1 4139
```

Note that the SELECT statement in the DEF of the ADMINS file being read or written is ignored by FACQUIR. Only the SELECT in the TAP file is effective.

17.3 TXTACQ: Acquire Text Files

The TXTACQ command is used to acquire text files into ADMINS files in a specific way. A TAP instruction file is not used. However the ADMINS file must contain an alphanumeric (An) field named LINE1.[4] For line of text in the file a record will be created in the ADMINS file with the first "n" characters of the text line being placed in the field LINE1. If the ADMINS file contains a second alphanumeric field named

4. LINE may be used in place of LINE1 for the name of the first field.

LINE2 then the characters that remain in each line in the file after LINE1 is filled up will be placed in the field LINE2. TXTACQ prompts for the names of the text file to be acquired and the ADMINS file which is to receive the text lines as records.

For example, assuming the following file has been defined:

```
*   TEXT.DEF
*
MAS 1000
*
SEQ I KEY1
...
LINE1 A80   "Columns 1-80 of the text line"
LINE2 A40   "Columns 81-120 of the text line"
...
```

and TXTACQ was run as follows:

```
$ txtacq
TEXT-CONTROL-FILE ADMINS-FILE:payroll.rep text.mas
42 RECORDS WRITTEN INTO TEXT.MAS
$
```

then each line of PAYROLL.REP would be placed in the fields LINE1 and LINE2 of TEXT.MAS. The TEXT-CONTROL-FILE and the ADMINS-FILE names may be included on the command line as follows:

```
$ txtacq payroll.rep text.mas
42 RECORDS WRITTEN INTO TEXT.MAS
$
```

TXTACQ can add records to a non-empty file and requests confirmation before doing so. Any response other then "Y" for yes will cause TXTACQ to terminate without processing the text file. For example:

```
$ txtacq
TEXT-CONTROL-FILE ADMINS-FILE:earnings.rep text.mas
TEXT.MAS ALREADY HAS 42 RECORDS, OK?y
38 RECORDS WRITTEN INTO TEXT.MAS
$
```

The text file input to TXTACQ can contain indirect references.[5]

17.4 FDATAP: Write External File

The contents of an ADMINS file can be written out to an external file usingFDATAP.

5. See Section 1.3.3 “Indirect References” for a general discussion of indirect references.

17.4.1 FDATAP: Write External Disk File

The FDATAP command is used to write an external file from an ADMINS file. The records per block (RPBLK) specification is always "1" in the file description line of a TAP file used with FDATAP. The dialogue of the FDATAP command is as follows:

```
$ fdatap
------.TAP OUTPUT-FILE-NAME
TYPE:text.tap text.fil
ADMINS DATA FILE NAME:status.mas
50 RECORDS WRITTEN INTO TEXT.FIL
$
```

The user provides the name of the "TAP" instruction file describing the format of the output records to be created by FDATAP. Then the user provides the name of the disk file into which the output records are to be written. FDATAP creates standard variable length records that are all of one size. FACQUIR can re-acquire files produced by FDATAP. When ASCII output is requested on the TAP file description line then any text editor can be used to read and edit the output file created by FDATAP.

17.4.2 FDATAP Field Description Options

FDATAP supports he following field description options:

17.4.2.1 Leading Zeroes In EN Fields

When a decimal or integer ADMINS field is written as an EN field the digits are right justified with leading blanks. If the field description has the option "Z" then the leading blanks will be leading zeroes. For example:

```
SALARY 24 10 EN Z
```

17.4.2.2 Overpunch (Minus) Sign In EN Fields

When a decimal or integer ADMINS field is written as an EN field the number is written as an absolute. If the field description has the option "O" then negative numbers will be written with an overpunch (or minus) sign. Options "Z" and "O" may be used together. For example:

```
BALANCE 47 10 EN Z O
```

17.4.2.3 Literals and Hexadecimal Constants

The user can also insert either literal data or a hexadecimal value anywhere in the output record. This is done by placing lines of the following format in the TAP file.

For literals:

```
- BPOS BLEN FORM ADMTYP STRING
```

A dash is used for the NAME specification. BPOS, BLEN, and FORM are the same as any other field description. ADMTYP is an ADMINS data type notation and STRING is the literal data. For example:

```
- 10 10 EN D2 9999.99
- 30 2 E A2 C3
```

The first example places the D2 value "9999.99" as an EBCDIC (or ASCII) numeric field starting at byte position 10 on the external record and extending 10 bytes long. The second example places the A2 string "C3" as an EBCDIC (or ASCII) two byte field starting at byte position 30 on the tape record.

For hexadecimal values:

```
HEX BPOS BLEN STRING
```

"HEX" is used as the NAME specification and BPOS and BLEN are the same as other field description lines. STRING is the hexadecimal digits to be placed in the record. For example:

```
HEX 30 2 C1F1
```

This example places the hexadecimal value "C1F1" in byte position 30 and 31 of the external record.

17.4.2.4 Conditional Hexadecimal Constant

FDATAP can be instructed to create a hexadecimal value in the output record conditionally on the value in an integer field in the ADMINS record. For example:

```
HEX 10 2 F1F2 COND
```

This will place the hex value F1F2 in byte positions 10 and 11 on the output tape record if the integer field COND is non-zero. Otherwise, byte positions 10 and 11 will be set to zero.

17.4.3 Select Via Key Range Option

FDATAP can be instructed to select only those records for output that fall within a specified key range. This is done by placing the word "KEY" followed by a low key(s) and a high key(s) in the TAP instruction file after the file description line. For example:

```
120 20
KEY 077385 888888
*
ACCT  1   6  EN
BAL   7  10  PD2
...
```

would write to tape only those records whose key was between 077835 and 888888. If the file has multiple keys then the key values that make up each field are separated by blanks in the instruction line. For example:

```
KEY 0100 4000 001 0100 4999 999
```

This illustrates three keys and the range is 0100-4000-001 to 0100-4999-999.

Because DATAP key range select uses the ADMINS direct access mechanism, the ADMINS data file should be in sort order for this feature to be correctly and reliably used.

17.5 IE: the ADMINS Import/Export Facility

IE (AdmIE), the ADMINS Import/Export facility, provides a generalized capability for transferring information between standard ASCII files and ADMINS data files. This facility is especially useful for passing information to and from the data interchange formats used by many popular desktop computer applications, such as spreadsheet and graphics applications.

By default the command:

```
$ ie n.mas
```

displays the contents of N.MAS on the standard output (e.g. the screen for an interactive terminal session).

IE options are specified using the following command line format:

```
$ IE -qualifier_1=(modifier_1,modifier_2) -qualifier_2 \
$         admins_datafile -qualifier_3
```

NOTE

Note: there should be no blanks on either side of the "=" in the IE command qualifiers.

Command qualifiers supported by the Import/Export facility are:

- COM - Read the command qualifiers and options for this run from the specified instruction (ASCII text) file. For example, if the file vsamp.ie has the following contents:[6]

  ```
  ! Sample IE instruction file
  vendor.mas
  -header
  -begin=++
  -delimiter=**
  -ending=&&
  -modify=(vname\s=10)
  !
  ```

  then the command:

  ```
  $ ie -com vsamp.ie
  ```

  will be equivalent to:

  ```
  $ ie vendor.mas -header -begin=++ -delimiter=** \
  -ending=&& -modify=(vname\s=10)
  ```

  If the COM qualifier is used no other qualifiers are allowed on the command line (the rest of the IE specification must be in the COM instruction file).
- ACQUIRE=ASCII_filename

  Read external data from an ASCII file into an ADMINS datafile.
- CREATE=ASCII_filename

  Output ADMINS data into an external ASCII file.
- DELIMIT=character(s)

  Use the given ASCII character(s) as a field delimiter.
- BEGIN=character(s)

  Insert the given ASCII character(s) at the start of each record.
- ENDING=character(s)

  Use the given ASCII character(s) as a record end delimiter.

6. Note that "!" delimits comments in IE instruction files.

You may use quotation marks to enclose the character strings specified by DELIMIT, BEGIN, and ENDING. Non-printing and context-sensitive characters are specified using the backslash[7] character (\), followed by the decimal ASCII code (see Appendix H.2 "Integer Decimal Values for ASCII Characters") for the character. For example, ENDING="\9\9" declares that two tab characters signal the end of a record.

- TABLE - The TABLE qualifier designates an ASCII file which contains a translation table to be used when exporting data (with CREATE) or acquiring data (with ACQUIRE). **The translation file is in effect only for alphanumeric fields** (An Fields).

  The format for entries in the translation table file is "map:dd1=dd2"[8]. That is, translate the character whose ASCII decimal code is "dd1" into the character whose ASCII decimal code is "dd2".

  This capability is especially useful for conversion of special 7-bit ASCII characters into their 8-bit versions and vice-versa. In the following example, the 7-bit codes for some special Norwegian/Danish characters are converted to 8-bit.

NOTE

```
! This is an example
! of a translation table.
!        1    1    2    2
!---5----0----5----0----5
! from -> to
!===========
map:123=230   ! converts "æ" to 8-bit
map:124=248   ! converts "ø" to 8-bit
map:125=229   ! converts "å" to 8-bit
map:91=198    ! converts "Æ" to 8-bit
map:92=216    ! converts "Ø" to 8-bit
 map: 93=197    ! converts "Å" to 8-bit
```

  A "*" or "!" in the first column of the translation table file signals a comment line.

- FIELDS=(field_1[\L=m][\NOCR][\S[IZE]=n],field_2[\L=m][\NOCR],[\S=n]...)

  By default, CREATE outputs all the fields in the ADMINS data file to the created ASCII file. Use the FIELDS qualifier to output a specific list of fields. The display width of fields in the list can be altered by appending the "\SIZE" field name qualifier. Output of text fields can be modified by appending the "\L" and "\NOCR" field name qualifiers, as described in Section 17.5.1 "Managing Text Fields in IE".

- MODIFY=(field_1[\L=m][\NOCR][\S[IZE]=n],field_2[\L=m][\NOCR],[\S=n]...)

  Use the MODIFY qualifier to alter the output of specific named fields,[9] while IE still outputs all the fields in the file. MODIFY supports the same field name qualifiers as FIELDS.

7. Backslash (\) and quotes (") are examples of context-sensitive characters. Backslash's special meaning is to indicate that what follows it is the decimal ASCII code for a character. Quotes are used to include blanks in the character string specification. To designate the backslash itself in a delimiter or ending you must use the decimal ASCII code for backslash (e.g. DELIMIT="\92"). To designate quotes as an delimiter use DELIMIT="\34".
8. For compatability with earlier versions of ADMINS, entries in the translation table may also be entered in the format map=dd1:dd2. However, this format is not supported in the REPORT environment file (REPORT$ENV) and the TRANS environment (TRANS$ENV). Using the recommended format map:dd1=dd2 allows you to use common tables for all three commands.

- HEADER - Use the datafile's fieldnames as a heading (with CREATE).
- LOTUS - Import or Export CSV-format ASCII text. Using the LOTUS (or CSV) quailifier sets up AdmIE to correctly handle CSV-format data. Specifyiing DELIMTER and ENDING is not necessary..

  With /CREATE: Output data in "CSV" format. Enclose the values in alphanumeric, picture, and date field types within double quotes. Use comma as a field delimiter. Suppress commas in numeric fields. Always uses "minus sign" for negative numbers (that is, logical name ADM$MINUS and setting "OPTION P" is ignored

  With /ACQUIR: Input ASCII file is in CSV format, first line must contain field names of ADMINS data file (delimited by commas).
- CSV - Same as LOTUS
- NOCRLF - (With CREATE) Do not append carriage return/line feed record terminator characters to the external record ending delimiter.
- NONULL - (With CREATE) Do not output zeroes for zero value numeric fields.
- NOPAD - (With CREATE) Squeeze out all trailing and leading blanks in fields being exported to an ASCII file.
- INSERT - (With ACQUIRE) Insert records read from the external ASCII file into the ADMINS datafile. (Records are appended by default, appending is a much faster operation.)
- NOBIN - (With ACQUIRE)
- LIST - Display the fields within the ADMINS datafile on the user's terminal.
- NOSTAR - Do not display processing progress graphics during external file acquisition/creation.
- SELECT="expression"

  Use the specified ADMINS SELECT statement to determine whether to accept the external record read into an ADMINS datafile (with ACQUIRE) or whether to output the current ADMINS record to the external ASCII file. The entire expression must be enclosed by double quote characters (").

All command qualifiers can be abbreviated to a minimum of two characters (SE for SELECT).

To load data into an ADMINS file, the first external record acquired specifies the record format, using the full field names from the ADMINS data file, and the previously specified field and record delimiters. Subsequent records in the external file are then loaded into the ADMINS file according to this specification. For example, given the following IE command line:

```
$ ie -acquire=mydat.dat mydat.mas -delimit="*" -ending="#"
```

where MYDAT.MAS has the following definition:

```
MAS 1000
DEPT     X9999 KEY1
REGION   A1
AGENT    X99
CURRENT  D
```

---

9. IE supports the D% or U% syntax in FIELDS and MODIFY lists for CREATE operations to reference the codelist description or user action code (see Appendix I.5.2.1) for a field i.e.: "FIELDS=(CUSTID,CUSTOMER,D%SALESREP,INVNO)", "MODIFY=(D%SALESREP\S=10)"

```
1PREV    D
2PREV    D
3PREV    D
4PREV    D
COMMENT A80
```

The beginning of the ASCII file to be acquired could appear as follows:

```
DEPT*REGION*AGENT*CURRENT*COMMENT#
2132*W*67*    2345*New hire took over for H. Hines #
2245*W*52*   67822*Was badly back ordered          #
1655*W*12*   22566*No new clientele                #
1654*C*22*   17881*Hard work starting to pay off   #
                ...
etc.
```

At least one delimiter character must separate each field, and at least one delimiter character (different from the field delimiter) must terminate each record.

When creating an external ASCII text file, users may use field and/or record delimiter characters, and external file fieldname header records optionally (as their application dictates). By default, IE outputs the following standard lengths for each ADMINS field type, without delimiters:

ADMINS field type	Default length	Examples
Alphanumeric (An)	(n//2) * 2	Rounded up to even number of chars. e.g: A16 = 16 chars., A5 = 6 chars.
Pictured (Xpic)	Pic length	XA9999 = 5 char.
Date (DA,DT)	ADM$DATE len	m d, Y4 = 18 char.
Time (TM)	11	
4-wd decimal (Fn)	36 + n	F2 = 38 char., F4 = 40 char.
Decimal (Dn)	30 + n	D = 30 char., D3 = 33 char.
Longword (Ln)	15 + n	L1 = 16 char
Integer (I)	7	
Internal Text (TI)	Document length	
External Text (TX)	Document file name length	

SELECT criteria are given using normal ADMINS expressions, and can be applied to either acquisition of external data, or creation of external files.

17.5.1 Managing Text Fields in IE

IE has special capabilities to aid in processing the transfer of information between external text files and ADMINS TI (internal text) fields.

- **CREATE:** To limit the length of internal text (i.e. TInn data type) output to the ASCII text file by CREATE, append "\L=m" to the TI field name, using either FIELDS or MODIFY. "\L=m" specifies that only the first "m" lines of the document contained in the TI field are to be output.

To control how CREATE interprets "hard" carriage returns[10] found in internal text documents created by ADMINS' TED text editor, assign the decimal ASCII code for the character you want IE to output when a hard carriage return is encountered to the logical name ADM_HARD_CR, e.g.:

```
> admlcr adm_hard_cr 10
```

10. "Hard" carriage returns result when a new line in the original document has been forced, for example by a carriage return keystroke. "Soft" carriage returns, by contrast, are the result of automatic line wrapping.

would output a "Line Feed" (or "ctrl/j") character (decimal ASCII code 10) to the external file whenever a hard carriage return is encountered in the internal text document. If the logical name ADM_HARD_CR is not assigned hard carriage returns in internal text are output as blanks. The effect of ADM_HARD_CR is global to all internal text fields being exported. To override this global effect for a particular field append "\NOCR" to the field's name in the FIELDS list, e.g.:

```
$ IE -CREATE=ASCII.TXT -FIELDS=(CUSTID,DELIVNOT\NOCR)
```

"\NOCR" means interpret a hard carriage return as a blank character.

- **ACQUIRE:** A new command line switch, -NOBIN can be used with AdmIE to discard (not acquire) "***\bin***"tagged binary data when acquiring RTF text.
- **ACQUIRE:** IE provides text pattern substitution for documents being acquired into internal text (TI) fields.

The character string patterns to be replaced are listed with their replacement strings in a text file named PATTERNS.TBL in the current default directory. (If PATTERNS.TBL is not found in the current default directory IE will attempt to translate the logical name ADM$PATTERN, which can be used to identify a directory specification where IE will look for PATTERNS.TBL.)

Three kinds of pattern substitution operations are supported in IE: deletion (discard the string), substitution (replace the string with another), and insertion (add something before or after the string).

Each entry[11] (line) in PATTERNS.TBL follows the general syntax:

```
MAP = "FIND-PATTERN" : "SUBSTITUTE-PATTERN"
```

Spaces are not required between each component, but may be used to increase legibility.

Each substitution consists of two distinct operations: finding a pattern in the text buffer; and replacing that pattern with a new pattern. Deletion is specified by a rule that contains only a FIND-PATTERN. FIND-PATTERN and SUBSTITUTE-PATTERN must be enclosed by double quotes (").

The backslash character (\) has a special function inside the quoted strings FIND-PATTERN and SUBSTITUTE-PATTERN: it signals that the special character that follows it is to be used literally and passed as part of the string in FIND-PATTERN and SUBSTITUTE-PATTERN. Some examples:

```
!Find and replace
MAP = " mispell " : " misspell "
!Find and delete.
MAP = " very,"
!Replace Bill in quotes with
!Bill in parentheses.
MAP = "\"Bill\"" : "(Bill)"
!Replace \eta\admnat with ETA:[ADMNAT]
MAP = "\\eta\\admnat" : "ETA:[ADMNAT]"
!Find and insert: "active" becomes "inactive"
MAP = "active " : "in&"
```

The last example demonstrates the use of the special "ditto" character, ampersand (&), in the SUBSTITUTE-PATTERN for insert operations. Ampersand is a place holder, representing the entire FIND-PATTERN transferred to the position of the ampersand in SUBSTITUTE-PATTERN. Some more examples:

11. Lines that begin with an asterisk (*) or exclamation point (!) are ignored, and may be used for comments.

```
!Find "a+b" and replace it by "(a+b)".
MAP = "a+b" : "(&)"
!
!Find " very" and replace it by " very, very".
MAP = " very" : "&,&"
```

To include a literal "&" in the SUBSTITUTE-PATTERN, precede it with a backslash. Ampersand has no special meaning in the FIND-PATTERN.

"Backslash" syntax can also be used to specify non-printing characters:

```
!Replace VT "bold on" escape sequence with TED's
MAP = "\27\[1m" : "\28\248"
!
!Replace VT "bold off" escape sequence with TED's
MAP = "\27\[0m" : "\28\240"
```

A numeric value following backslash is interpreted as the decimal ASCII code for a character.

IE loads the input text into an internal buffer for processing the substitution rules. When the buffer is full, the substitution rules are applied, in order, on the contents of the buffer. Then the buffer is output to the TI field, where it is formatted according to that field's ruler (see Appendix I.7.2 “Overview of Internal Codelist Table Values ”).

Multiple-word FIND-PATTERNs are not supported.[12] (FIND-PATTERN should have no imbedded blanks or tabs. Blanks or tabs should only occur at the beginning or the end of the FIND-PATTERN).

Patterns are case sensitive. There is no limit to the number of substitution rules that can be placed in "PATTERNS.TBL". The text being acquired is changed according to the first rule. The output of the first rule becomes the input text for the second rule, and so on until there are no more rules.

ACQUIRE supports special "meta character" syntax to search for patterns that match classes of characters or text of indefinite length.

?	Question mark matches any single character. The pattern X?Y matches XAY, X+Y or X?Y.
[ ]	Square brackets mean that the characters following "[", up to the next "]", form a character class, i.e. any single character from the bracketed list is considered a match. The pattern [aA] matches a or A.
-	In a bracketed string the dash indicates "through notation", i.e. [a-z] means "from a to z", or "match any lowercase letter".
^	Circumflex negates the character class that follows it in a bracketed string. [^a] means "not a" or "match any character except a.

12. "Words" in the input text (i.e. a character string delimited by blank, tab, or carriage return/line feed) are never split between buffers. But because consecutive "words" may be split by the buffering process - before the FIND-PATTERN is applied, FIND-PATTERNs that contain imbedded blanks or tabs cannot be found reliably in the input and consequently should not be used. When the text is placed in the buffer, blanks and tabs are preserved but CR/LF is converted to a blank. For consistency, a FIND-PATTERN that begins with a blank, e.g. " Hello" will match the first word of a document that begins: "Hello. Its me! My name is..." even though the document does not begin with a blank.

Use backslash before the character if you want to include ^, -, [, or] explicitly in a class.

*	Any text pattern followed by an asterisk (*) means match zero or more successive occurrences the pattern. For example, a* matches zero or more a's; aa* matches one or more a's; [a-z]* matches any string of zero or more lower case letters.

17.5.1.1 Initialization File for Acquiring Internal Text

AdmIE can use an initialization file when acquiring text into internal text fields that have been declared as RTF fields. To set this up:

1. Use AdmTed to create an RTF file with the fonts, tab settings etc. you want for the acquired text, and then type

   %%MYTEXT%%
   into the file and save it.
2. Assign the logical name

   ADM_RTF_INITFILE
   to point to the file you have just created (e.g. MY_DIR:MyInitfile.rtf)
3. run AdmIE to acquire the text.

The internal text field will contain the acquired text, formatted according the selections you made in the ADM_RTF_INITFILE "initialization file".

Chapter 18:ADED: The Data File Editor

ADED is an editor for ADMINS data files which can be used from **any ASCII terminal,** i.e. both a CRT or a hard copy unit.

18.1 Function of ADED

1. Append or insert new records into a file.
2. Delete individual records in a file.
3. Search out records in a file by their key value, or by selection criteria based on comparing values in fields to constant values.
4. Change values in fields in the file.
5. Display either all fields or groups of fields from one or many records in a file. A columnar format is used when it "fits" within the display width, or else the detail of a record is spread across several lines.

18.2 ADED Dialogue

When ADED is called it prompts for a file name.

```
$ aded
TYPE "HE" FOR HELP
FILE-NAME:
```

If the user types "HE" to the first FILE-NAME prompt, ADED displays a brief summary of all the ADED instructions and their use. When the help display is completed, ADED will re-prompt for the FILE-NAME. When the user types a file name, the file is opened by ADED, the fields in the file are displayed, the "record pointer" is placed at the first record of the file, the prompt "TYPE:" appears, and the user may begin editing.

The "record pointer" refers to the location, i.e. the specific record, in the file where ADED will begin processing the next instruction.

The user may type the file name directly on the command line if desired. For example:

```
$ aded pers.mas
PERS.MAS OPENED, 50 RECORDS
SS#/K1  LNAME  ADDRESS  BIRTHDA
***** TOP OF FILE
TYPE:
```

18.3 ADED Instructions

The ADED editing instructions that may be entered in response to the "TYPE:" prompt are as follows.

18.3.1 HEADING

```
TYPE:hx field1 field2 etc.
```

The HEADING instruction is used to associate fields names with a "heading number". For example, "H3 LNAME ADDR BIR", would set heading number "3" to the fields LNAME, ADDRESS, and BIRTHDA. As we can see a partial letter match is sufficient to place a field name in a heading. From then on whenever the user refers to heading "3" ADED will understand it is to use these fields. (Key fields are automatically included in each heading. Heading "0", which consists only of the key fields, is automatically set by ADED when the file is opened, and is used whenever the user doesn't specify any heading.)

18.3.2 PRINT

```
TYPE:p[x] [n]
```

The PRINT instruction displays "N" records from the record pointer according to the optional heading "X". The record pointer is left at the last record displayed. If "N" is absent it is assumed to be 1. If the page width is adequate then the data is displayed in columnar format under field name headings. Otherwise, the field names and the data are run together on as many output lines as are necessary to contain the requested information. In "brief" mode the field names are not displayed.

For example, "P3 2" in verify mode would display the following:

```
SS#------  LNAME----  ADDRESS---------  BIRTHDA--
059381528  JONES      3 ELM STREET      06-MAY-46
047317665  SMITH      146 MAIN STREET   14-JUL-34
```

18.3.3 WIDTH

```
TYPE:w n
```

The WIDTH instruction sets the page width. The maximum width is 128 characters. The initial width is 80 characters.

18.3.4 VERIFY

TYPE:ve

The VERIFY instruction sets the brief option off. In brief mode ADED does not display the field names associated with the data.

18.3.5 BRIEF

TYPE:br

The BRIEF instruction turns the brief option on which suppresses the display of field names. Initially the brief option is off.

18.3.6 TOP

TYPE:t

The TOP instruction moves the record pointer to the top of the file, that is to the first record in the file.

18.3.7 END

TYPE:e

The END instruction moves the record pointer to the end of the file, that is to the last record in the file.

18.3.8 NEXT

TYPE:n[x] [n]

The NEXT instruction moves the record pointer forward "N" records and then displays the current record using optional heading "X". "N" is optional and assumed 1 if absent.

18.3.9 UP

TYPE:u[x] [n]

The UP instruction moves the record pointer backwards "N" records and then displays the current record using the optional heading "X". "N" is optional and assumed 1 if absent.

18.3.10 FIND

```
TYPE:f[x] [key values]
```

The FIND instruction is used to find a record via its key values. Those key values not supplied on the instruction line by the user are requested by prompts from ADED. If a record is found it is displayed according to optional heading "X" and the record pointer is left at the found record. The search is applied across the whole file. For example, "F3 059361528" in brief mode would display the following:

```
059361528  JONES      3 ELM STREET      06-MAY-46
```

18.3.11 LOCATE

The LOCATE instruction is used to locate records based on any value in the record. The syntax of LOCATE is:

```
TYPE:l[x] [n]
SELECTION:field op cons
SELECTION:field op cons
SELECTION:cr
```

This means locate "N" instances that satisfy the selection criteria and display using the optional heading "X". "N" is optional and assumed 1 if absent. If "N" is "0" the search limit is not for a specific number of instances, but rather for all matches until end of file is reached. The record pointer is left at the last record located.

The selection criteria is a relation between a named field and a constant. Multiple criteria are "and-ed" together, i.e., they must **all** be true to locate a record. The relational operators are "EQ LE GE NE GT LT IN". IN is for alphanumeric data, i.e. LNAME IN DAVID, for LNAME includes "DAVID", e.g. there is a match when the string "DAVID" is present in the field LNAME.

18.3.12 CHANGE

```
TYPE:c field value [n]
```

The CHANGE instruction is used to change the content of fields in the record. The syntax means to take VALUE and put it in FIELD for "N" records. The record pointer is left at the last record changed. If "N" is absent it is assumed to be 1. Both key fields and non-key fields may be changed. However, while the key fields may be changed, the index structure is not modified to reflect that change, and the user must SORT the file after exiting from ADED.

18.3.13 APPEND

```
TYPE:a[x]
```

The APPEND instruction puts ADED in a mode where it will append records to the end of the file until append mode is left. ADED will prompt the user to type values for the fields in the records to be appended. Only those fields in heading "X" will be prompted. However, if no heading number is given then ADED prompts for **all** the fields in the record. The user leaves append mode by typing carriage return to a prompt for the first key field. The record pointer is left at the last record appended.

18.3.14 INSERT

```
TYPE:i[x] [key values]
```

The INSERT instruction proceeds like the FIND instruction, accepting key values from the user and trying to find the record with those key values. If a record is not found, then the INSERT instruction behaves like the APPEND instruction, except that INSERT does not prompt for key values, i.e. it uses the key values which it searched on. Also, INSERT inserts the record in sort order, whereas APPEND adds to the end of the file. ADED only inserts one record at a time, that is after the values are supplied for the new record, ADED asks for another instruction. The record pointer is left at the inserted record.

18.3.15 DELETE

```
TYPE:de [key values]
```

or

```
TYPE:de this
```

The DELETE instruction proceeds like a FIND instruction only after the record is found it is deleted. If a record is not found with the key values then nothing is done, and the record pointer is placed at the first record in the file.

In the second variation "DE THIS" will delete the record at the current record pointer location. In either variation after a record deletion the record pointer is left at the record following the one that was deleted.

18.3.16 CLOSE

```
TYPE:cl
```

The CLOSE instruction closes the active file and prompts for a new file name. The only proper way to leave ADED is to close the active file, and type carriage return to the next "FILE NAME:" prompt.

18.4 ADED Example

The following example illustrates the result of using all of the ADED instructions. A simple file with only 10 records is used. The contents of the entire file are shown at the beginning and the end of the ADED editing session.

```
$ aded emp.mas
EMP.MAS OPENED, 10 RECORDS
EMP#/K1  FNAME  SEX  HIRE  STATUS
***** TOP OF FILE
TYPE:h1 fname sex hire status
TYPE:p1 15
EMP#  FNAME-------  SE  HIRE-----  ST
0013  DANIEL        M   29-SEP-80  P
0204  JACK          M   24-FEB-75  P
0293  GERALD        M   16-MAY-68  P
0425  BARBARA       F   17-MAY-82  P
0445  JEANNE        F   23-JUN-80  P
0593  RAY           M   16-MAR-65  P
0843  ALLEN         M   23-MAY-73  P
0924  SUSAN         F   07-SEP-82  P
1012  VERNON        M   10-SEP-76  P
1213  CYNTHIA       F   29-APR-77  P
***** END OF FILE
TYPE:t
***** TOP OF FILE
TYPE:w 30
TYPE:p1 2
EMP#: 0013  FNAME: DANIEL
SEX: M  HIRE: 29-SEP-80
STATUS: P
--------------------
EMP#: 0204  FNAME: JACK
SEX: M  HIRE: 24-FEB-75
STATUS: P
--------------------
TYPE:w 80
TYPE:br
TYPE:p1
0204  JACK          M   24-FEB-75  P
TYPE:ve
TYPE:p1
EMP#  FNAME-------  SE  HIRE-----  ST
0204  JACK          M   24-FEB-75  P
TYPE:t
***** TOP OF FILE
TYPE:p1
EMP#  FNAME-------  SE  HIRE-----  ST
0013  DANIEL        M   29-SEP-80  P
TYPE:e
***** END OF FILE
TYPE:p1
EMP#  FNAME-------  SE  HIRE-----  ST
1213  CYNTHIA       F   29-APR-77  P
TYPE:u1 5
EMP#  FNAME-------  SE  HIRE-----  ST
0445  JEANNE        F   23-JUN-80  P
TYPE:n 3
EMP#
0924
TYPE:f1
EMP#:0445
EMP#  FNAME-------  SE  HIRE-----  ST
0445  JEANNE        F   23-JUN-80  P
TYPE:l1 1
SELECTION:hire lt 01-jan-66
SELECTION:cr
EMP#  FNAME-------  SE  HIRE-----  ST
0593  RAY           M   16-MAR-65  P
TYPE:c status s 2
```

```
STATUS: FROM "P" TO "S"
STATUS: FROM "P" TO "S"
TYPE:u
EMP#
0593
TYPE:p1 2
EMP#  FNAME-------  SE  HIRE-----  ST
0593  RAY           M   16-MAR-65  S
0843  ALLEN         M   23-MAY-73  S
TYPE:i1 0234
FNAME:robert
SEX:m
HIRE:01-NOV-82
STATUS:p
RECORD INSERTED
TYPE:a1
EMP#:0678
FNAME:mary
SEX:f
HIRE:15-DEC-82
STATUS:p
RECORD APPENDED
EMP#:cr
***** END OF FILE
TYPE:de 0445
RECORD DELETED
TYPE:p1
EMP#  FNAME-------  SE  HIRE-----  ST
0593  RAY           M   16-MAR-65  S
TYPE:de this
RECORD DELETED
TYPE:t
***** TOP OF FILE
TYPE:p1 15
EMP#  FNAME-------  SE  HIRE-----  ST
0013  DANIEL        M   29-SEP-80  P
0204  JACK          M   24-FEB-75  P
0234  ROBERT        M   01-NOV-82  P
0293  GERALD        M   16-MAY-68  P
0425  BARBARA       F   17-MAY-82  P
0843  ALLEN         M   23-MAY-73  S
0924  SUSAN         F   07-SEP-82  P
1012  VERNON        M   10-SEP-76  P
1213  CYNTHIA       F   29-APR-77  P
0678  MARY          F   15-DEC-82  P
***** END OF FILE
TYPE:cl
EMP.MAS CLOSED, 10 RECORDS
FILE-NAME:cr
$
```

18.5 Restricting Use of ADED

All use of ADED can be restricted by placing "G" in the logical name OPTION. ADED can be restricted to read-only operations by placing "H" in the logical name OPTION. That is, the insert, append, delete and change operations are inhibited. Both options are described in Appendix A: "Options". ADED checks the system and group logical name tables **first** for the presence of the "G" or "H" before the process table is used. Hence the ADED availability for the entire system or an entire group can be controlled because "GRPNAM" and "SYSNAM" privilege are needed to change logical names in the group and system tables, respectively.

ADED can also be restricted to operate on only one file each time ADED is invoked. If there is a lowercase "a" in the logical name OPTION ADED will exit when the file it is operating on is closed (using the ADED "CL" command), without prompting for another file name. This feature allows ADED use within command procedures to be restricted to a specific file in situations where file and data security are important issues.

Chapter 19:Concurrency Control: Multi-User Files

ADMINS Concurrency Control (the multi-user file facility) supports shared use of ADMINS data files by multiple users.

19.1 Modes of File Access

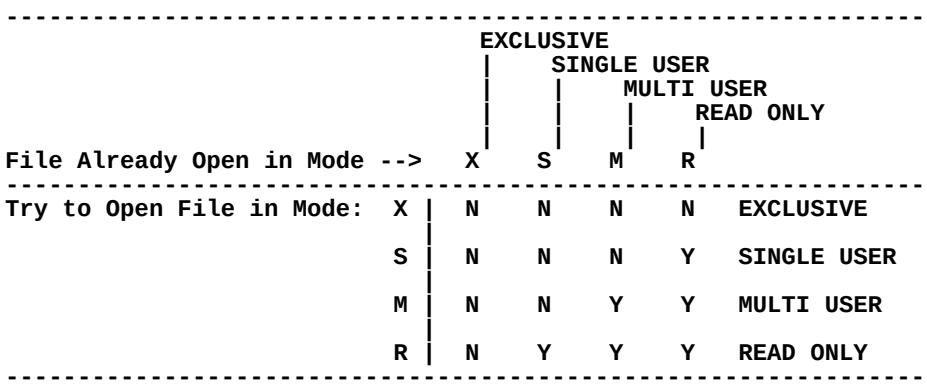
ADMINS commands all utilize the same concurrency control system when they read and write records in ADMINS data files. This makes it possible for any combination of ADMINS commands (for example, TRANS and PROD) to obtain fully coordinated concurrent access to the same file. Normally, though, a command such as PROD is the only "user" of its files.

There are four basic modes in which a file can be processed.

- **EXCLUSIVE mode (X):** While a user has EXCLUSIVE use of a file, no other user can open it for ANY purpose. If a file is already in use, it cannot be opened for EXCLUSIVE use.
- **SINGLE USER mode (S):** While a user has a file open for a SINGLE USER activity, no other user can open the file for writing (other users can open the file for READ ONLY use). If a file is already open for anything except READ ONLY use, it cannot be opened SINGLE USER.
- **MULTI USER mode (M):** Provides automatic record locking, block overwrite protection, and file index coordination. Any number of users can open a file in MULTI USER mode, and can perform any operations on the file concurrently. If a file is already open in EXCLUSIVE or SINGLE USER mode it cannot be opened MULTI USER.
- **READ ONLY mode (R):** READ ONLY mode provides the ability to read any[1] file, unless it is open for EXCLUSIVE use.

1. ADMINS data files that contain text fields have two files associated with them, the text storage file (TSF) and the text catalog file (TCF), as described in Appendix K.1. Users must have write access to these files in order to access the ADMINS data file that contains text fields.

The compatibility among the four modes can be summarized as:



Each ADMINS command has default file processing modes for the various files it uses. The default modes provide the appropriate and most efficient level of concurrency control for typical uses of the command.

Command	File	Default File Processing Mode
ACQUIR/ FACQUIR	OUTPUT	Exclusive
ADED	MAIN FILE	Single User
ANALYZER	ACTIVE, LINK	Read Only
AV	MAIN FILE	Multi User if write, otherwise Read Only
AFU	MAIN FILE	Exclusive if INIT or RESTORE, otherwise Read Only
CMP	All	Read Only
DATAP/FDATAP	INPUT	Single User
IE	MAIN FILE with /CREATE with/ACQUIRE	 Read Only Exclusive
MAINT	MAIN FILE OUTPUT	Single User Exclusive
MOVE	INPUT LINK LINK WRITE ADD OUTPUT	Single User Read Only Single User Single User Exclusive
MRGFIL	INPUT OUTPUT	Single User Exclusive
PROD	DETAIL LOOKUP OUTPUT	Single User Single User Exclusive
REPORT	All	Read Only
SCREEN	All	Read Only

Command	File	Default File Processing Mode
SORT	SELF-SORT	Exclusive
	INPUT	Single User
	OUTPUT	Exclusive
TRANS	ACTIVE	Multi-User
	ACTIVE NOSHARE	Single User
	FIELD LOG	same as active file
	LINK	Multi User Read (RM)
	LINK W	Multi User
	APPEND	Multi User
	INDEX	Multi User
	LOOKUP	Read Only
SUBROUTINES	that open an ADMINS file	
	open the file	Read Only
	except...	
	OUTPUT	Exclusive

In general, except for LINKs without writeback and LOOKUP, TRANS opens files MULTI USER by default. The "batch" processing commands generally open input files SINGLE USER and always open output files EXCLUSIVE.

These defaults provide automatic protection of data integrity. They simplify application development by reducing the need for developers to decide how files should be opened. The defaults provide optimum performance consistent with the degree of concurrency control usually needed by each ADMINS command.

19.1.1 Overriding the Default File Processing Mode

In some applications the default file processing mode may not give the desired type of protection. In those situations, the default file processing mode can be overridden by appending the codes (-X, -M, -S, -R) to the file name. EXCLUSIVE mode cannot be overridden when a file is being initialized (e.g. by AFU INIT or self-SORT, MOVE and SORT can also initialize the output file).

Three flexible file processing modes are available as options on a per-file basis (-RM, -SM, and -RX). When ADMINS commands read from the disk, whole "blocks" of data, which might contain several records, are read into memory. A command that is accessing a file read-only (-R) file physically re-reads the disk only if it needs a block not currently in memory, because read-only file access implies that other concurrent users of the file will not be changing it, or that any such changes do not impact this file access.

If this assumption is not valid, use "-RM" to instruct ADMINS to open the file for READ ONLY processing, but to check for updates (i.e. always re-read the record from the disk), and check for new record insertions or deletions (i.e. check for changes to the index). Like "-R", "-RM" access never locks the file, so it is compatible with all the same concurrent accesses as "-R". "-RM" file access is recommended for situations where concurrent users may add records or alter records in the file, but compatibility with concurrent users precludes file locking.

However, because the file is never locked from being changed by another concurrent file access it is possible for "-RM" to try to access the file while it is in the midst of processing a change (insertion, deletion) initiated by another concurrent access. When this happens the "-RM" file access may exit with an error condition. Use file locking (-RX file access) to prevent this.

If read-only access is desired, but concurrent file accesses are likely to add or delete records in this file and cause error exits, use "-RX" to instruct ADMINS to use file locking. Locking ensures that the processing of changes initiated by other concurrent accesses will be complete. "-RX" file access does lock the file for brief periods while searching the file's index and reading the record, so it has the same compatibility with other concurrent uses as "-M".

The "-RM" and "-RX" modes of file access take more time than read-only ("-R") access, due to the additional checking and disk i/o that is taking place.

"-SM" tells ADMINS to try to open the file SINGLE USER; but, if it cannot be opened that way, try to open it MULTI USER.

19.2 Resolving File Access Conflicts

The following options control the action of ADMINS when a file cannot be opened because it is already open in a conflicting mode:

Option	Meaning	Default in:
A	Prompt User: Wait or Exit	--
W	Wait for file to become available	Batch
E	Exit with error message	Interactive

To change the default file opening action globally, the option letter (A, W, or E) is assigned to the logical name ADM$FILEOPTION in the system logical name table.

To override the default file opening action for a specific use of a file, append the appropriate letter to the file name (-A, -W, -E). A file opening action code can be combined with the optional file processing codes above. For example, STUFF.MAS-XW would tell ADMINS to open STUFF.MAS in EXCLUSIVE mode and, if that is not possible, wait until the other users leave the file and it can be opened for EXCLUSIVE use.

19.3 Resolving Record Access Conflicts

The following options control how ADMINS behaves when a user is locked out of a record because another user currently has write access to the record.

Option	Meaning	Default in:
L	Wait until record is available	Batch
I	Ignore record lock	--
N	Prompt user: Wait or Ignore	Interactive
T	Do not lock top record in file (on a per file basis only)	

When Record locking option "N" is in effect and the target record is not available the user is prompted "Record Lockout on <Filename> Do you want to Wait or Ignore"; i.e. do you want to wait for the record to become available, or, do you want to ignore record lockout and access the record for read-only use[2]. If the choice is to wait for the record, and the record does not become available; ADMINS will prompt with the same choice every ten seconds.

Record locking option "T" keeps the first record in the file free of locks, to accommodate applications that use this record as a special location[3], i.e. MATCH or F$F in TRANS; or use a two-record file to maintain sequential numbers or running totals (one record to maintain values, the other a "dummy" record to give applications "a place to land"). Option T is only available on a per file basis, i.e. XXX.MAS-T.

Record locking option "I" is provided only as an escape hatch for older applications which used the now obsolete record lockout (RLKOUT) feature, in case record locking causes problems in a specific application. Record locking option "I" bypasses the record locking facility in the sense that the user does not receive a message or a prompt if a record is locked, and the user can read the record. However, **ADMINS will not write back the record.** When option "I" is invoked, screens which use the old RLKOUT functionality will work exactly as they used to.

The RMO can detect when record locks are ignored, either as a result of record locking option "I" or because the user has typed "I" to the "Do you want to Wait or Ignore" prompt. This facility is discussed in Section 16.21 "Managing Ignored Record Locks - ADM$NOLOCK and ADM$NLREC".

To override the default record locking action globally, assign the appropriate code to the logical name ADM$FILEOPTION.[4]

To override the default record lockout action for a specific use of a file, append the code to the file name (STUFF.MAS-MWL tells ADMINS to try to open STUFF.MAS MULTI USER, wait until it can be opened that way, and, whenever a locked record is encountered, wait for it to become available).

2.If the TRANS Environment File (see Section 6.17.13.1 "TRANS main program") contains the entry "recordlock.ask=once" then once a recordlock is ignored TRANS will enter "read-only" mode and remain in that mode until a new virtual record becomes active and no record locks are being ignored

3."-T" should only be used to give applications a "safe" place they will not be locked out of. ADMINS will not write the a record in a file that is not locked because it is the first record in a file opened using "-T"

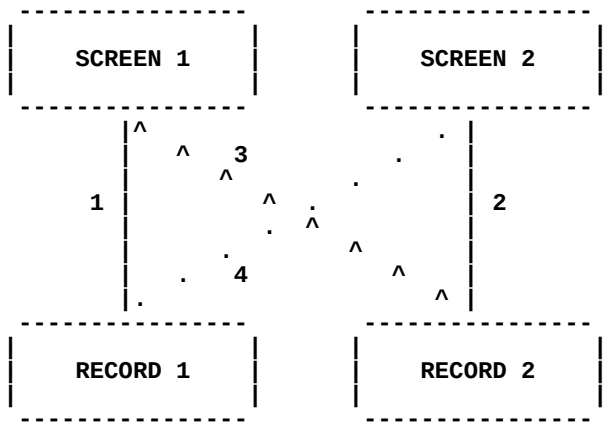
4. Option T "Do not lock the top record in the file" is available only on a per file basis, i.e. it cannot be enabled using ADM$FILEOPTION.

19.3.1 Special Treatment of Full-Block Records

ADMINS takes advantage if multi-user files are designed so that multiple users cannot simultaneously update the same disk block. If the record size is padded[5] to **exactly 512 words** (1024 bytes - the size of an ADMINS disk block) then ADMINS does not need to re-read the disk block to check for someone else's updates to other records in the block before writing the block back to disk, because the record being written is the only one in the block. This can cut down disk i/o by up to 33%, and cut down processing time in batch-type operations that update records with the file open multi-user.

19.4 Resolution of Deadlocks

ADMINS concurrency control automatically handles deadlocks. A deadlock situation can be visualized as follows:



The screen in this example has an active file and a link, and it can write back to both the active record and the link record. SCREEN 1 has RECORD 1 as its active record and SCREEN 2 has RECORD 2 as its active record. The screens each obtain locks on their active records without difficulty (Steps 1 and 2). Then (Step 3), SCREEN 1 tries to link to RECORD 2, is locked out of the record, and begins waiting for it to become free. In Step 4, SCREEN 2 tries to link to RECORD 1, is locked out, and begins waiting for record 1. Since each screen is waiting for the other one to complete a transaction, they will wait forever unless something is done.

Deadlocks are rare events which never happen in most applications. When they do occur, however, ADMINS concurrency control performs the following actions designed to protect data integrity:

1. First, the Lock Manager detects the deadlock and chooses one of the deadlocked users as a "victim".
2. TRANS notifies the victim that the screen is deadlocked and that it will pause before the user can continue the transaction.

---

5. Pad the file by adding fields to the DEF. Each A80 field adds 40 words to the record size. When AFU reports Record size is "512W" the file size is just right.

3. If the victim has made any changes in editable fields which have not yet been written to the disk, TRANS updates the disk files.
4. TRANS releases all the victim's locks, submits new requests for locks on the same records, and starts to wait.
5. The other user obtains access to the record he was previously locked out of, and completes the transaction.
6. The ex-victim's pending lock requests are granted, all records in the screen are re-read from the disk, the screen is refreshed, and the ex-victim can complete the transaction.

Thus, ADMINS handles this difficult situation as smoothly as possible from the point of view of preserving the integrity of the data. If additional processing is required before the "victim" screen releases its locks or after it re-reads the disk, the RMO can obtain control. If the special field L$OCK/A4 is present as a local field in the RMO, the following occurs in the victim's TRANS during a deadlock:

```
------------------------------------------------
                           RMO calls:
                           S$S     M$M   L$OCK
------------------------------------------------
Normal pre-link RMO call   Field   UX    (blank)
Link results in deadlock
Deadlock message
Pre-release RMO call       Field   UX    DLOC
Write back all records
Release record locks
Queue requests for records
Wait until records are available
Re-Read records
Post-deadlock RMO call     Field   UX    WAKE
Normal post-link RMO call  Field   UP    (blank)
Refresh screen
------------------------------------------------
```

If L$OCK is in the RMO there is a pair of special RMO calls which occur only in a deadlock. Before TRANS writes back records because of a deadlock, TRANS calls the RMO with L$OCK set to 'DLOC'. When the records have been re-read from the disk, the second special call occurs, with L$OCK set to 'WAKE'. S$S has the usual value and M$M is set to the pre-link call.

For example, before writing back, the values of certain fields might be saved in local fields; and after re-reading the records, the fields might be compared with the saved values and action taken if the fields were changed by the other user.[6]

It should be emphasized that deadlocks will only present problems in a few applications. In most of those, the automatic deadlock handling provided by ADMINS will suffice to maintain data integrity. We have provided the hooks to the RMO for any applications where developers want additional control over deadlock processing.

Finally, if deadlocks occur in an application but for some reason they are truly not important, record locking can be disabled for the specific LINK W which causes the deadlock by using the (-I) record locking option after the link file name.

6. At these special deadlock calls, the RMO should not try to make TRANS leave the record. TRANS will not leave the record, because the transaction is not complete.

19.5 MLOCK: Lock Monitor Utility

Use MLOCK, the ADMINS Lock Monitor Utility, to monitor record and file locking. MLOCK has several command line qualifiers:

Qualifier	Function
s	Sort MLOCK report by Process ID
c	Continuous display (stop with Ctrl/C)
r (seconds)	Refresh interval for continuous display (default = 5)
a	Display multi-user file header detail information (i.e. last record, last index pointer, index root)
i (PID)	Display locks for specified Process ID
g (GID)	Display locks for specified Group ID
f (filename)	Display locks for specified filename[a]
u (username)	Display locks for specified user.
t (termname)	Display locks for specified terminal.
l (logid)	(UNIX only) Display locks for the specified value of ADM$LOGICAL
tex[t] (string)	Text substitution facility[b]
h	Help

a. Selection by file specification (f) and user name (u) support partial string matching. For example, "$ mlock -u fr" displays lock information for usernames FRED, FREDRICK, FRANK, FRMIS, etc.

b. See Section 19.5.2 "MLOCK Output Text Formatting Facility"

Combinations of qualifiers are logically AND-ed, i.e.

```
$ mlock -u fred -f vendor.mas   (UNIX)
```

displays locks for vendor.mas held by username fred.

19.5.1 MLOCK Output Display

The following shows a typical MLOCK output listing:

```
   __PID Logid Username_______ Comd Hostname________ Resource____ Mode Que_
File: c:\AppMenu\TEST_D~1\ADM_DD~1.ADD
000080d0   123 bd                  Uranus           RO File      Null Grnt
File: c:\AppMenu\TEST_D~1\ADM_DD~4.ADD
000080d0   123 bd                  Uranus           RO File      Null Grnt
File: c:\AppMenu\TEST_D~1\ADD314~1.ADD
000080d0   123 bd                  Uranus           RO File      Null Grnt
File: c:\AppMenu\TEST_D~1\ADM_DD~2.ADD
000080d0   123 bd                  Uranus           RO File      Null Grnt
File: c:\AppMenu\TEST_D~1\ORDER.MAS
000080d0   123 bd                  Uranus           MU File      Null Grnt
                                                 Rec 49       Writ Grnt
File: c:\AppMenu\TEST_D~1\CUSTOMER.MAS
000080d0   123 bd                  Uranus           RO File      Null Grnt
```

Locked Resource ("Resource") could be one of the following:

```
MU File (Multi-user file lock)
SU File (Single user file lock)
EX File (Exclusive file lock)
RO File (Read only file lock)
Rec nnn (Record lock)Multiuser File
```

Lock queue ("Que") could be either

```
Grnt (Granted)
Conv(ert)
 Wait
```

The lock mode codes ("Mode") are:

```
Null
Writ(e)
Excl(usive)
```

19.5.2 MLOCK Output Text Formatting Facility

To customize the output MLOCK produces for file locks use the TEXT qualifier:

`mlock -tex[t] ["]character_string["]`When MLOCK is called with the TEXT qualifier it displays "character_string" for each file lock. Special tokens may be inserted into "character_string" to identify the points where MLOCK should substitute specified values from the file lock. These special tokens are:

Token	MLOCK will replace with
%pid	Process ID that holds file lock
%username	Username that holds file lock
%file	Name of locked file

The tokens can be abbreviated two characters, i.e. "%f" for "%file" and "%u" for "%username". If "character_string" contains embedded blanks it must be enclosed in double quotes.

The c (continuous) qualifier is disabled when the text qualifier is used.

For example, the command:

```
$ mlock -text "%u has %f locked"
```

could be used to generate output that looks like this:

```
bd has /home/acctg/june/jrnl.mas locked
bd has /home/acctg/june/vendor.mas locked
bd has /home/acctg/june/inv.mas locked
randy has /home/acctg/history/sales.mas locked
randy has /home/acctg/history/cust.mas locked
```

Redirecting formatted MLOCK output text to a file can be useful for application monitoring and communicating with users. The output file could be used directly as a all or part of a command file or script.

Chapter 20:

(This chapter's content only appears in the OpenVMS version of the Manual.)

Chapter 21:Printer Queues

ADMINS commands use the operating system's printer queuing (or "spooling") facilities to handle printed output.

Printable output of ADMINS commands is placed in a file named ADMINSxx.LIS in the user's default directory (where "xx" is a unique automatically-generated number[1]). The ADMINSxx.LIS file is then placed on a print queue where the physical printing of the file is controlled by the operating system's queuing facilities.

Application users need not learn the inner workings of queuing. However, a working knowledge of how to create and control print queues is essential for the **system manager.**

21.1 ADM$SPOOLn: Logical Print Queue Specification

The output file ("ADMINSxx.LIS") may be queued to any print queue. Print queues are identified by number (0 through 255 with 0 the default) in the various ADMINS syntaxes described below. ADMINS commands append this number to the string "ADM$SPOOL" and then use the result as a logical name that identifies the destination print queue.

For example, if report output is routed to "queue 7" via REPORT's LP statement:

```
LP 1 7
```

then the output will be printed on the queue assigned to the logical name ADM$SPOOL7.

There may be any number of print queues assigned to "ADM$SPOOLn" logical names. For example:

To assign logical queue 0 to server “ourserver’s” printer hp1 and logical queue 26 to “ourserver’s” hp3

```
$ admlcr adm_spool0    \\ourserver\hp1
$ admlcr adm_spool26   \\ourserver\hp3
```

The queue number is set by the respective ADMINS command as outlined below:

1. REPORT uses the second element of the "LP" statement to set the queue number as described in Section 7.17.8.2 “Logical Queuing Device Number”.
2. TRANS uses the SPn keyword on the screen header line to set the queue number as described in Section 5.3.1.9 “SPn or TTn: Print Device Specification”.

---

1. On Windows the name of the most recently generated ADMINSxx.LIS file is placed in the logical name adm_listfile

3. ANALYZER uses OPTION SP to set the queue number as described in Section 12.19.10 “Spooler Number”.

21.2 Number of Copies Specification

The various ADMINS command syntaxes described below are used to specify a number of original copies (maximum=99) of the ADMIINSxx.LIS file to be produced by the print job. As stated above, if the COPIES qualifier appears in the ADM$SPOOLn logical name the number of copies specified there overrides the number that appears in the command syntax.

1. REPORT uses the first element of the "LP" statement to request a number of copies as described in Section 7.17.8.1 “Multiple Copies”.
2. ANALYZER uses OPTION NC to request a number of copies as described in Section 12.19.11 “Number of Copies”.

21.3 Deleting the Output File (ADMINSxx.LIS)

If the letter "D" is included in the logical name OPTION (see Appendix A: “Options”) ADMINS will request deletion of the ADMINSxx.LIS file after it is printed.

If OPTION "D" is not implemented some provision must be taken by the system administrator or the user to delete the ADMINSxx.LIS files on a regular basis.

21.4 Output to Non-queued device, ADM$PRT0

In some cases the user may wish send output directly to a physical device, without creating a ".LIS" file and without using the system print queue. To do this, assign the name of the device to the logical name ADM$PRT0.

Output is sent directly to ADM$PRT0 if the OUTPUT TT0 statement is used in REPORT, as described in Section 7.17.7.6 “OUTPUT TT0 (Direct Output to Physical Device)”, or if the TT0 keyword is present on the screen header line, as described in Section 5.3.1.9 “SPn or TTn: Print Device Specification”.

For example, if a report contained OUTPUT TT0 and the following assignment existed, the output would be directed to terminal _TTB3:.

```
$ assign _ttb3: adm$prt0    (OpenVMS syntax)
```

Appendix A:Options

The logical name OPTION may be assigned with up to a 24 character text string. Each character of the text string selects a specific optional behavior in ADMINS.

For example, the following assignment of the logical name OPTION will cause screens generated via ctrl/p to be immediately spooled for printing (S), and disables automatic file enlargement of Level 2 files(9).

```
> admlcr OPTION S9
```

The brief descriptions of each option that follow include cross references to sections of this manual where you can find more information on their context and function.

NOTE

ADMINS recommends that "L" **always** be included in OPTION.

Character	Function
* (asterisk)	Line of asterisks to show progress through a file is suppressed in all ADMINS "batch" commands.
, (comma)	Suppresses commas when TRANS is displaying numeric fields.
0	Zeroes in numeric fields are displayed as blanks as described in Section 2.4.2.1 "Input and Output Representation Options".
5	The string %CR% is used to represent a "hard carriage return" in response to a parameterized <<prompt>>, instead of the string CR. This is for situations where CR may be a valid response to a prompt, and therefore cannot be used in a command file to indicate that the response to a substitutable parameter's prompt should be a "carriage return." Note that when used in command files, 5 must be present in the string assigned to the logical name option both when the command file is "compiled" (NATCOM.EXE) and when the resulting COMxxx.COM is run.
6	Modifies the function of the Y$EAR subroutine (see Appendix H.4.5 "Y$EAR - Extracting the Year from a Date") so that it returns a value equal to the year of the date given less 1900, i.e. YEAR = Y$EAR(DATE) where DATE is set to July 23, 2004 will load the value "104" into YEAR. Ordinarily (without 6 in OPTION), the value "2004" would be loaded into YEAR.
\	Prevent string in angle brackets from being evaluated as substitutable parameter as described in Section 14.3.6 "Disabling Parameter Parsing: Treat <string> As Normal Characters".
7	Enables a special RMO call in TRANS. If the user (or SETKEY) types CTRL/L before entry into a field, TRANS calls the RMO with S$S set to the field the cursor is on, and M$M set to 'XX'. This feature can be used, for example, to trigger special RMO processing regardless of where the cursor is.

Character	Function
9	Disables automatic file enlargement (see Section 1.8 "Dynamic Data File Expansion").
D	Delete the spool (.LIS) file after it has been printed. Section 21.3 "Deleting the Output File (ADMINSxx.LIS)" describes the use of the spool file and the delete and save option.
E	Use the "European" character set option, i.e. treat the characters "[", "]", and "\" as ordinary characters. Therefore, in TRANS the KEY2 and KEY3 functions are disabled (see Section 6.5) and the "\" character is replaced by the "PF2" keystroke as the ERR keystroke (see Section 6.8 "Control Functions").
F	Accept "-" as a numeric character in a "9" (digit) position in a picture field as described in Section 2.4.2 "Field Data Types".
G	Disables ADED (see Section 18.5 "Restricting Use of ADED"). (The system logical name table is searched first, then the group logical name table, and finally the process logical name table.)
H	ADED (see Section 18.5 "Restricting Use of ADED") is usable only in read-only mode. (The system logical name table is searched first, then the group logical name table, and finally the process logical name table.)
J	Accept input with reversal of the comma and decimal point as used in Europe. This is applicable to integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types as described in Section 2.4.2.1 "Input and Output Representation Options".
K	Output with the comma and decimal point reversed as used in Europe. This is applicable to integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types as described in Section 2.4.2.1 "Input and Output Representation Options".
L	Do not reset other "K" or "KC" fields when a LINK causes a CHECK statement in TRANS to evaluate to "true" as described in Section 5.4.1 "LINK Paragraph". This is an optional fix due to the possible effect on existing application instruction files.
N	Aggregation control in SORT is controlled only by the KEY fields as described in Section 2.4.4.1 "Method of Operation". Normally, aggregation is controlled by the KEY fields and the ASC/DESC fields.

Character	Function
P	Use parentheses for minus values in output presentation. This is applicable to integer (I), longword decimal (Ln), decimal (Dn), and four-word decimal (Fn) field types as described in Section 2.4.2.1 "Input and Output Representation Options". This setting is IGNORED by AdmReport when it is creating CSV or Excel XML format data (see Section 7.24 "Report Command Line Options")
T	Suppresses automatic left justification of characters in alphanumeric fields in TRANS. Leading blanks are not eliminated on input as described in Section 5.6 "Screen Layout".
a	Enables ADED to restrict users to a specific file. ADED (see Section 18.5 "Restricting Use of ADED") only opens one file. The file name must be on the command line; the command exits when that file is closed by the user.
b	When LFEXIT is active, the NEXT keystroke which files the record causes TRANS to continue displaying that record instead of advancing to the next record as described in Section 6.3.1.1 "Update Mode Under LFEXIT Control". This OPTION setting is re-evaluated by TRANS after the first BEGREC RMO calls which occur when a screen is entered.
c	The echoing of prompt strings and parameter values for logical (L_) parameters are suppressed. If a logical parameter is assigned, then NATCOM (see Section 14.3.3 "Logical Parameters"), CMP (see Section 9.7.1 "Logical Parameters"), DEFINE (see Section 2.11.1 "Logical Parameters"), and REPORT (see Section 7.14.2 "Logical Parameters") do not display the prompt string or the value assigned.
d	Tells REPORT to create the special field TODAY as field type DT rather than field type DA, which is the default. "d" must be present in OPTION at compile time if you want to use TODAY/DT in a pre-compiled REPORT (RPO).
f	Suppress formfeed at end of REPORT output.
k	Changes the default action of TRANS when a partial or non-existent key value is entered in update mode. When this option is in effect, TRANS goes to the first record whose key is greater than the one which was entered, instead of the last record whose key is less (see Section 6.6 "Record Moving and Searching").
o	Suppresses the printing of any HEADING or SUMMARY when a REPORT SELECT OR KEY statement results in no records being processed.
p	Enables the use of parameterization in SCREEN (see Section 5.17 "Parameterization").

Character	Function
r	Enable REPORT RETRY by turning off automatic deletion of RPxx.TMP file as described in Section 7.14.4 “Rerunning Parameterized Reports, RETRY”.
s	Re-read link directly from disk rather than buffer whenever KC or C field is entered, whether or not the KC or C field is changed. This OPTION setting is re-evaluated by TRANS after the first BEGREC RMO calls which occur when a screen is entered.
t	Retranslate ADM$DATE at every display field as described in Section 2.4.2 “Field Data Types”.
u	DEFINE will create files (by default) in the user's current default disk and directory. Normally DEFINE (by default) creates the file in the same disk/directory location as the DEF file. (Of course, any logical name or device name explicitly identified in the DEF itself ALWAYS takes precedence in determining the location of the output file). See Section 2.3.3.1 “Utilizing DEFs in Other Directories”.
x	Suppresses the display of picture fields with null values (blanks and zeroes).
y	All commands give a "Record Lockout" message, as TRANS does, when they are waiting for a locked record. By default, or if record locking option 'L' is in use, all commands (TRANS excepted) wait for a locked record with no message. This option has no effect in batch mode.
z	Suppress formfeed at beginning of REPORT output.
~	Enable enhanced menu bar capability when compiling TRO in SCREEN. See Section 5.12.3 “Enhanced Accelerator Capability”.

Appendix B:Special Logical Names used by ADMINS

The following logical names have special meanings and/or functions in ADMINS.

Logical Name and Function	Reference
A$<string> Communication with AV command	Section 13.6 "AdmAv - Communicate with ADMINS Files via Logical Names"
A$<string> TRANS command line arguments	Section 6.15 "Entering TRANS On A Specific Record"
A$AV_ERR Set to error status when AV error exits	Section 13.6 "AdmAv - Communicate with ADMINS Files via Logical Names"
ADM$APPLIC Used with OPTION M to explicitly specify string to be placed in the process name.	Appendix A: "Options".
ADM$CENTURY_CUTOFF_YEAR Allows you to change how values entered with two digit years into date fields are interpreted and stored.	Section 2.4.2 "Field Data Types"
ADM$COLLATE Collating Table identifier	Section 2.12 "Alternative Collating Sequences"
ADM$COLLDIR Collating Table directory identifier	Section 2.12 "Alternative Collating Sequences"
ADM$COMMANDS Identifies a text file that contains 'symbol definitions' to be used when ADMINS generates a PERL script from an ADMINS command file.	Appendix C.2 "Commands and Procedures"
ADM$COM_STARTUP Identifies a PERL script that executes just prior to executing the actual content of the ADMINS command file, and may be used to: log startup information like the value of logical names, start time, user name, etc	Section 14.15 "Localized Command File Extensions"
ADM$COM_NORMALEXIT Identifies a PERL script that executes if the PERL script exits normally (at the EOC: label)	Section 14.15 "Localized Command File Extensions"

Logical Name and Function	Reference
ADM$COM_ABNORMALEXIT Identifies a PERL script that executes if the PERL script exits abnormally (at the EOC4: label)	Section 14.15 "Localized Command File Extensions"
ADM$DATE Date format	Section 2.4.2 "Field Data Types"
ADM$DD Data Dictionary data file directory	Appendix I.12.1 "Logical Names and Symbols "
ADM$DD_CHECK_DESCR Converting applications to ADD	Appendix I.13.1 "Converting Data Files "
ADM$DD_DIST Data Dictionary application directory	Appendix I.12.1 "Logical Names and Symbols "
ADM$DD_FILE Producing DEF files from Data Dictionary	Appendix I.11.1 "Data Dictionary Reports "
ADM$DD_FILEDEF Producing DEF files from Data Dictionary	Appendix I.11.1 "Data Dictionary Reports "
ADM$DD_LOAD Converting applications to ADD	Appendix I.13.1 "Converting Data Files "
ADM$DD_LOAD_SOURCE_DESCR Converting applications to ADD	Appendix I.13.1 "Converting Data Files "
ADM$DIALOGBOX AdmCom and AdmReport prompt for parameters in a Dialog Box rather than in the command prompt window (if set to the value "P").	Section 1.3.4 "Parameterization"
ADM$DIR_LOGEVENTS Log occurrence of certain events.	Section 1.12 "Logging Events and Fatal Errors"
ADM$DIR_LOGFATAL	Section 1.12 "Logging Events and Fatal Errors"
ADM$DIST ADMINS commands directory identifier	Chapter 1: "Introduction"
ADM$EMSG ADMINS diagnostic messages directory identifier	Section 1.11.1 "Operation of the Message Facility"
ADM$FILEOPTION Global file access method control	Section 19.2 "Resolving File Access Conflicts"
Case sensitivity in path specification (UNIX only)	Appendix C.1 "File and Device Specification Differences"

Logical Name and Function	Reference
ADM$FORMAT Report automatic formatting file identifier	Section 7.20 "Data Description File for Automatic Formatting"
ADM$G0_ONLY Select Graphic Character Set by Alternate Method	
ADM$GBL Identifies directory for use with GBLSTORE subroutine	Appendix H.13.8 "GBLSTORE - Access TRANS Global Area on Disk"
ADM$KB_TYPE Override TRANS environment type	Section 6.17 "The TRANS Environment File"
ADM$HARD_CR Control IE output of input hard CR	Section 17.5.1 "Managing Text Fields in IE"
ADM$HELPDIR TRANS help file directory identifier	Section 6.14 "HELP in TRANS"
ADM$HELPFILE TRANS help file identifier	Section 6.14 "HELP in TRANS"
ADM$KEY_ESC Allows you to escape from a key sequence.	Section 6.6 "Record Moving and Searching"
ADM$LEVEL Controls diagnostic message level	Section 1.11.2 "Expanded Message Facility"
ADM$LISTFILE (UNIX-only) Name given to most recent "ADMINSxx.LIS"	Appendix C.1.1 "Differences in Print File and Temporary File Naming"
ADM$LNKMEM Control maximum file size for ANALYZER LINK MEM	Section 12.3.2 "Efficiency Considerations"
ADM$LOCALE Localizing messages and prompts	Section 1.9 "Localizing ADMINS"
ADM$LOGFILE Interactive command dialogue log file identifier	Section 1.5 "Logging Interactive Sessions"
ADM$MAGTAP Magnetic tape device identifier	Chapter 17: "External Data Files"
ADM$MASK_LOGEVENTS Specify events to log	Section 1.12 "Logging Events and Fatal Errors"
ADM$MAX_FIELDS Control maximum number of fields in TRANS	Appendix F.2 "SORT"
ADM$MINUS Control indicator for negative number	Section 2.4.2.1 "Input and Output Representation Options"

Logical Name and Function	Reference
ADM$OBJECT Object files directory identifier, SCREEN	Chapter 5: "AdmScreen: Compiling Screen Forms"
ADM$OBJECT Object files directory identifier, REPORT	Section 7.21 "Pre-Compiled Reports"
ADM$OBJECT Object files directory identifier, CMP	Chapter 9: "CMP: The Record Maintenance Compiler"
ADM$OUTPUT_RFM Specify alternate record format for REPORT output file.	Section 7.17.7.1 "OUTPUT LP (Line Printer)"
ADM$PATTERN IE pattern substitution table directory identifier	Section 17.5.1 "Managing Text Fields in IE"
ADM$PRT<n> Bypass spooling/queuing	Section 21.4 "Output to Non-queued device, ADM$PRT0"
ADM$READONLY Make all TRANS file opens read-only	Section 6.4 "Entering or Changing Fields"
ADM$REPSRT Control max number of records for SORT in REPORT	Section 7.12.2 "Comparison of SORT Statement and SORT Command"
ADM$SCRIPT Name given by to most recent script file created by COM	Appendix C.1.1 "Differences in Print File and Temporary File Naming"
ADM$SCRNAM Name of currently active TRANS screen	Section 5.5.8.7 "ADM$SCRNAM"
ADM$SCR_VIDEO Video attributes	Section 5.10 "Video Highlighting Facilities"
ADM$SPOOL<n> Print queue identifier	Section 21.1 "ADM$SPOOLn: Logical Print Queue Specification"
ADM$SRTMP SORT temp working file directory identifier	Section 4.2.1 "Temporary Files"
ADM$SRTOUT SORT temp output file directory identifier	Section 4.2.1 "Temporary Files"
ADM$STYLE Style table identifier - REPORT	Section 7.17.3 "LENGTH Statement"
ADM$TERM Logical name usually loaded with terminal identifier	Appendix C.1.1 "Differences in Print File and Temporary File Naming"

Logical Name and Function	Reference
ADM$TEST_TODAY Creates "testing value" for TODAY and NOW.	Section 5.5.8.1 "TODAY: Current Date"
ADM$TEXTEDIT Identifies/enables VIEWTEXT editor	Appendix H.6.9 "VIEWTEXT: Display Text File in TRANS"
ADM$TRANS_MESSAGE Status line message	Section 6.13 "Status line"
ADM$TRANS_VIDEO Run-time video attributes	Section 5.10 "Video Highlighting Facilities"
ADM$TRONAM Name of currently active TRO	Section 5.5.8.8 "ADM$TRONAM"
ADM$TX_DEFAULT External text file directory identifier	Appendix K.1 "Special Considerations"
ADM$TX_DIRECTORIES External text file directories list identifier	Appendix K.1 "Special Considerations"
ADM$VTLEN Control TRANS display size	Section 5.6 "Screen Layout"
BRIEF Control terminal dialogue in command files	Section 14.9 "BRIEF and VERIFY"
KEY$<string> Value for REPORT KEY statement	Section 7.13.3 "KEY Statement"
L$<string> Logical parameters	Section 1.3.4 "Parameterization"
L$<string> Communication with AV command	Section 13.6 "AdmAv - Communicate with ADMINS Files via Logical Names"
L$<string>MAINT Key range using logical names	Section 10.3.1 "Operate on Key Range"
L$<string>PROD Key range using logical names	Section 11.2.1 "PROD with Key Range: PROD/KEY"
L$AV_ERR Set to error status code when AV error exits	Section 13.6 "AdmAv - Communicate with ADMINS Files via Logical Names"
OPTION Set ADMINS optional functionality	Appendix A: "Options"
PERL_OPTION Wait for RETURN after abnormal termination (if set to "W")	Section 14.16 "PERL_OPTION Logical Name: Pause Before Exit at Abnormal Termination"

Logical Name and Function	Reference
REPORT$ENV REPORT environment file identifier	Section 7.22 "The REPORT Environment File"
TED$ENV TED environment file identifier	Appendix J.6 "The TED.ENV File"
TPR$ENV TPR environment file identifier	Appendix J.12.1 "The TPR.ENV Environment File"
TPR$FIELD Specify field for TPR/INT	Appendix J.12 "Printing Text Fields: TPR"
TPR$FILENAME Specify file for TPR/INT	Appendix J.12 "Printing Text Fields: TPR"
TPR$FROM_PAGE Specify starting page for TPR/INT	Appendix J.12 "Printing Text Fields: TPR"
TPR$KEY<n> Specify key value for TPR/INT	Appendix J.12 "Printing Text Fields: TPR"
TPR$TO_PAGE Specify last page for TPR/INT	Appendix J.12 "Printing Text Fields: TPR"
TRANS$ENV TRANS environment file identifier	Section 6.17 "The TRANS Environment File"

Appendix C:Platform and Operating System Differences

ADMINS provides an environment for building, supporting and running applications that is consistent, compatible, and highly portable across all supported hardware platforms and operating systems. Usually, instructions (e.g. source code, command procedures, syntax) for application modules can be utilized without change in any supported environment. When instructions must vary depending on the operating environment, conditional compilation instructions can be imbedded in the source code, so that only a single copy of the source need be maintained.

The sections that follow describe the areas where ADMINS applications are or may be affected when moving to a different operating system or the hardware platform.

C.1 File and Device Specification Differences

ADMINS handles path specifications (file or directory specifications) in the same way wherever they appear, i.e. the specification is converted to all lowercase characters (case insensitive) and handed over directly to the operating system, except that on Win32 systems if the path specification character string begins with a logical name, it is translated by the ADMINS logical name server, and the translation is substituted into the string before the string is handed to the operating system.

If WORK_DISK is a logical name and you type the following command at the system prompt in any supported environment:

```
$ trans work_disk:demo.mas
```

you begin a TRANS General Editor Mode session on the file DEMO.MAS, which is located in the directory assigned to the logical name WORK_DISK. Similarly this same file is specified in any supported environment by a REPORT or RMS instruction file that includes the statement:

```
FILE WORK_DISK:DEMO.MAS
```

Path specifications strings (after logical name substitution on Win32 systems) must result in a valid path or file specification in the host environment.

Devices are usually specified in ADMINS via the use of logical names, e.g. the direct print device is identified the logical name ADM$PRT0, and the default print queue is identified by the logical name ADM$SPOOL0. Any device identifier string you use with ADMINS, whether directly or by assigning it a logical name, must be a valid device name in the host environment.

C.1.1 Differences in Print File and Temporary File Naming

Some ADMINS commands create temporary files which they utilize during their processing, e.g. SORT creates a temporary work file with a name in the form SORTxx.TMP and COM creates a host operating system command file with a name in the form COMxx.

ADMINS commands that format and queue files for printing automatically name the print file with a name in the form ADMINSxx.LIS.

On OpenVMS systems the contents of the logical name ADM$TERM is used to automatically name these files, i.e. SORTA2.TMP and ADMINSA2.LIS are created if the value A2 is assigned to the logical name ADM$TERM.[1]

On Win32 systems the ADMINS lock manager server process ensures that each one of these temporary files gets a unique name when it is created.[2] To provide access to automatically-named shell procedures created by COM, the server loads the logical name adm_script with the name of the most recently created script file. To provide access to automatically named print files the server loads the logical name adm_listfile with the name of the most recently created "ADMINSxx.LIS" file.

For example, if after running a report you want to merge the output into another document, you can find out the name given to the output file by translating adm_listfile after running the REPORT.

C.2 Commands and Procedures

Many ADMINS commands accept arguments and/or qualifiers on the command line (if required arguments are not given on the command line the command will prompt for them).

The method for expressing positional and/or keyword arguments for a particular command does not vary when the host environment changes, e.g. the following commands are valid either for OpenVMS or for Win32.

```
$ cmp journal

$ trans vendor.tab 132 insert
```

The method for expressing command line qualifiers does vary depending on the host environment. On Win32 systems, command line qualifiers are delimited by the "-" or "/" characters preceded by a space, while on OpenVMS systems command line

1. OpenVMS distributions of ADMINS include a command procedure, "ADMTERM.COM", that can be called at login to automatically assign a value to ADM$TERM. This procedure assigns the unique portion of the device name to ADM$TERM for interactive sessions and the last three digits of the job name to ADM$TERM for batch sessions.
2. Unique names are required on Win32 systems because the Win32 file system is "versionless". If a file is created with the same pathname (full file specification) as an existing file the new file will over-write the old file. On OpenVMS systems the new file would be created with a new version number, the old file would persist.

qualifiers are delimited by the "/" character (no preceding space is required). The following examples show equivalent commands in the OpenVMS and Win32 environments:

```
$ move/v                ! OpenVMS MOVE/Virtual
> move -v               # Win32 MOVE/Virtual

$ define/redef vendor   ! OpenVMS Re-Define file
> define /redef vendor  # Win32 Re-Define file
```

ADMINS commands may be run in batch procedures or perl scripts on Win32 and in DCL command procedures on OpenVMS. These procedures have entirely different syntaxes, so they are not portable between the two environments.

ADMINS provides a highly portable alternative to direct coding of command and shell procedures, the ADMINS command procedure pre-processor, which is described in Chapter 14: "Command Files". The pre-processor reads an "ADMINS command file" and translates it into a procedure appropriate for the operating environment[3]. For example the following simple ADMINS command file runs a report:

```
display running demo report...
report demo
display demo report complete
```

On OpenVMS NATCOM translates this command file into the following command procedure

```
$ Q :==
$ ADM$EXIT_SEVERITY :==
$ ON ERROR THEN GOTO EOC4
$ASSIGN/nolog N BRIEF
$WRITE SYS$OUTPUT " COMA2.COM   STARTED"
$SHOW TIME
$WRITE SYS$OUTPUT " ----------------------------"
$WRITE ADM$OUTPUT "running demo report..."
$IF Q THEN GOTO EOC2
$report demo
$WRITE ADM$OUTPUT "demo report complete"
$EOC:
$ RE$START :==
$WRITE SYS$OUTPUT " ----------------------------"
$WRITE SYS$OUTPUT " COMA2.COM   TERMINATED"
$GOTO EOC1
$EOC2:
$WRITE SYS$OUTPUT " -------------------------------"
$WRITE SYS$OUTPUT " QUIT FROM COMA2.COM  "
$GOTO EOC1
$EOC4:
$ ADM$EXIT_SEVERITY :== '$SEVERITY
$WRITE SYS$OUTPUT " -------------------------------"
$WRITE SYS$OUTPUT " ABNORMAL TERMINATION COMA2.COM  "
$WRITE ADM$OUTPUT " ABNORMAL TERMINATION COMA2.COM  "
$EOC1:
$SHOW TIME
```

3. On Win32 systems a perl script is generated and executed. On OpenVMS systems a DCL command file is generated and executed.

On Win32 Adm2Perl translates this command file into the following shell procedure.

```
     use Cwd;
use File::Basename;
fileparse_set_fstype("MSWin32");
$ENV{'ADM_INCOM'}="Y";
$exit_sev='ADM_EXIT_SEVERITY';
$adm_basename = "showacf";
$fatal_exit=1;
system("AdmLcr $exit_sev 0");
($adm_cwd = cwd()) =~ tr/\//\\/;
$last = substr($adm_cwd, -1, 1);
if ($last eq '\\') { chop $adm_cwd; }
system("AdmLcr ADM_SCRPATH $adm_cwd\\A0020e40.pl");
$dttm = localtime(time());
$startmsg = "A0020e40.pl started $dttm";
$| = 1;
print " $startmsg\n";
print(" ----------------------------\n");
print ("running demo report...\n");
system('report demo ');
$cur_exit = $? >> 8;
if ($cur_exit == $fatal_exit) {
   system("AdmLcr $exit_sev $cur_exit");
   goto EOC4;
   }
print ("demo report complete\n");
EOC:
print(" ----------------------------\n");
print("A0020e40.pl TERMINATED\n");
$dttm = localtime(time()); print $dttm, "\n";
system('admldl RESTART_0020e40');
system("AdmDel ADM_SCRPATH");
exit 0;
EOC4:
print(" ----------------------------\n");
print("ABNORMAL TERMINATION A0020e40.pl\n");
$dttm = localtime(time()); print $dttm, "\n";
$cur_exit = `AdmLtr $exit_sev`;
exit $cur_exit;
```

The two procedures listed above are very different, but the results the user sees when the ADMINS command file is run in either host environment is quite similar:

```
$ COM DEMO            ! on OpenVMS

READING DEMO.COM
COMA2.COM   WRITTEN

COMA2.COM   STARTED
28-MAY-1993 10:26:09
----------------------------
running demo report...
REPORT DEMO
demo report complete
----------------------------
COMA2.COM   TERMINATED
28-MAY-1993 10:26:20
```

```
com demo                    !On Win32
Reading showacf.acf
A0020e57.pl written
A0020e57.pl started Fri Mar 21 12:47:56 2003
----------------------------
running demo report...
REPORT demo
demo report complete
----------------------------
A0020e57.pl TERMINATED
Fri Mar 21 12:47:57 2003
```

C.3 Concurrency Control and Network Access

Concurrent access to ADMINS data files and individual records in ADMINS data is essentially the same in either the OpenVMS or the Win32 environments. On OpenVMS the "distributed lock manager" is used by ADMINS to control concurrent file and record access. For Win32 systems ADMINS provides a "lock manager" server[4] to provide similar control of file and record access.

On OpenVMS systems concurrency control is provided for network access (read or write) via either VMS clusters[5] or DECnet.

On Win32 TCP/IP networks concurrency control is provided for network access (read or write) for all client Win32 nodes that request file access via the same ADMINS Win32 lock manager server process..

See Chapter 19: "Concurrency Control: Multi-User Files" for details.

C.4 ADMINS Logical Names for Win32

Logical names are used in many different ways to communicate with and/or control the operation of ADMINS commands. A logical name can be thought of as a "container" for a text string.

Logical names are a built-in facility of the OpenVMS operating system, and are commonly created, managed, and utilized by OpenVMS (DCL) operating system commands outside of ADMINS. On Win32 systems, ADMINS provides logical name capability, but **only ADMINS commands can utilize or manipulate logical names.**

ADMINS Logical Names for Win32 provides an OpenVMS-like logical name[6] facility for use in ADMINS-based applications running in the Win32 environment. This service is accessible from ADMINS screens, reports, rmos, etc., as well as on the command line. Logical Name Server commands provide for the creation, modification, translation and deletion of logical names.

---

4. The ADMINS Win32 lock manager server must be running in order to use ADMINS (admsv is usually found in the ADM$DIST directory). Usually the ADMINS lock manager server runs as a service on one of the nodes in a network, and the other nodes designate that node as their lock manager server node.
5. VMS clusters are so closely coupled that for ADMINS purposes they may be considered a single system.
6. ADMINS Logical Names for Win32 was developed to provide a compatible environment for ADMINS applications originally developed on OpenVMS. However, the logical name concept provides an easy-to-use vehicle for two way communication between parent and child processes that developers will find many uses for beyond the maintenance of compatible applications. Two way communication means that if a child modifies a logical name, the parent process has access to this new value. This is important and useful because, unlike VMS, Win32 creates a new process for each command/program that is run, e.g. TRANS is run by a child process of the (shell) process that called it.

Use the following commands to manipulate logical names in the ADMINS Logical Name Server.

C.4.1 Create/Modify a Logical Name

admlcr creates or modifies logical name.

Usage:

```
admlcr [-t table] logical value.
```

Examples:

```
> admlcr adm_dist c:\progra~1\admins\bin

> admlcr -td mydir c:\myfiles

> admlcr –ts our_root \\homenode\share1
```

- **-t** Table in which logical name is to be created/modified:
  - — ***s*** system
  - — ***d*** desktop
  - — ***p*** process (default)

    For example, to add a logical name to your desktop table use –td.

    The desktop table differs from the process table in that it will persist when the user logs off or the computer is shut down.

    The system table is maintained by the lock manager server process. Entries in this table are available to all client processes that subscribe to the same lock manager server. Only privileged users can change the contents of the system table.
- **logical** The logical name to be created/modified. This string is always converted to all upper case characters before it is placed in the table. To ease porting OpenVMS applications all dollar signs ($) found in the logical name are converted to underscores (_) before the name is placed in the table.
- **value** The value to be assigned to the logical name.

C.4.1.1 Create multiple logical names using a file

```
admlcr @FILENAME
```

assigns multiple logical names at once. FILENAME is the name of a text file that contains lines in the following format:[7]

```
[-tTable] LOGICAL_NAME  Value
```

for example;

```
-tP mydir C:\myfiles
-tP adm_commands mydir:mycommands.fil
-tP adm_debug_rmo Y
-tP adm_object mydir:
```

The table argument is optional, entries are placed in the process table by default.

In this file tokens in the form t***%ENVVARIABLE%*** can be used to place environment variable values into logical names. E.g.

```
L_USRNAM %USERNAME%
```

7. Appendix C.4.4 “Show All Logicals” describes a method for generating files in this format from the current logical name environment.

will pick up the environment value for USERNAME and assign it to the logical name L_USRNAM (in the process table).

C.4.1.2 Special Syntax for logical name OPTION

If the logical name being assigned is OPTION and the value contains a two character string that starts with "+" or "-" the character following "+" or "-" will be added to or removed from the value of the logical name OPTION. E.g.

```
AdmLcr option -5
```

will remove "5" from the current value of OPTION in the process table, and

```
AdmLcr -td option +f
```

will add "f" to the value of OPTION in the desktop table.

C.4.2 Translate a Logical Name

admltr prints the translation of the specified logical name to the standard output.

Usage:

```
admltr [-t table] [-f ] logical
```

Examples:

```
> admltr ADM_DIST
c:/progra~1/admins/bin

$temp=`admltr adm_dist` ;

print "admins directory is $temp\n" ;

! in a perl script!
admins directory is c:/progra~1/admins/bin


> admltr -f mydir
t:\ourgroup\harry
```

- **-t** Table in which logical name is to be create/modified:
  - — ***s*** system
  - — ***d*** desktop
  - — ***p*** process (default)
  - — ***e*** displays environment variable

  For example, to add a logical name to your desktop table use –td.

Nothing is printed if there is no translation.

- **-f** Specifies full iterative translation when translation itself contains logical names.

Examples:

Given the following logical name assignments:

```
admlsh
ours  t:\ourgroup
mydir ours:harry

admltr –f mydir
t:\ourgroup\harry
```

C.4.3 Delete a Logical Name

admldl removes the specified logical name from the specified logical name table.

Usage:

```
admldl [-t table] logical
    > admldl tempdir
```

The logical name specified may contain a wildcard character, for example:

```
admldl -td *
```

would delete all the logical names in the desktop table, while:

```
admldl L_SPEC_*
```

would delete all logical names in the process table that begin with the string "L_SPEC_"

By default, admldl removes logicals from the Process Table.

- **-t** Table in which logical name is to be create/modified:
  - ***s*** system ! requires privilege!
  - ***d*** desktop
  - ***p*** process (default)

C.4.4 Show All Logicals

admlsh shows all the logical names in the the specified logical name table(s). (All tables by default).

Usage:

```
admlsh [-t table] [-L] [-f] [string]
```

Examples:

```
> admlsh

> admlsh -td X_
```

If the optional string is specified, only logical names that begin with the string are listed (case insensitive).

- **-f** Specifies full iterative translation when the translation itself contains logical names.
- **-L** will cause the listing to be in the form:

  ```
  -tTable LOGICAL_NAME  Value
  ```

  Sending this output to a file, e.g.

  ```
  AdmLsh -L > LOG.FIL
  ```

  can the be used to recreate the logical names using:

  ```
  AdmLcr @LOG.FIL
  ```

  as described in Appendix C.4.1.1 "Create multiple logical names using a file"

C.4.5 Managing Process Logical Names

The ADMINS logical name server provides 3 tables of logicals: system, desktop, and process. While the system logical name table translates from VMS quite well, the Win32 process and desktop logical name tables are a little different. Unlike VMS, Win32 creates a new process for each command/program that is run in a session. For compatibility, each of these "child" processes of the same "parent" login session access the same logical name table, which we call the "process" table, although it is actually a "login session" logical name table. The desktop table is very similar to the process table, except that its contents will still be there the next time you login, or if you shut the computer off and turn it on again.

ADMINS can use the **environmental variable** ADM_LOGICAL to identify a family of processes that are part of the same session. That is you can set the environment variable adm_logical to cause subsequent commands in that session to use a different process logical name table.

For example:

```
> admltr mydir
  c:\myfiles

> set adm_logical 77

> admltr mydir
  d:\alt_files
```

ADM_LOGICAL can be set to any value between 0 and 99. If the environment variable is not set ADMINS will act as if it is set to 0.

C.5 Setting up ADMINS for the User

On OpenVMS ADMINS commands are called using a symbolic name, e.g. "TRANS" is usually the symbolic name given to the executable image "ADM$DIST:TRANS.EXE". The symbolic names that are commonly used for ADMINS commands are listed in Appendix C.5.1 “OpenVMS Symbols for ADMINS commands”.

On Win32, when ADMINS commands are called they are found via the PATH environmental variable, which contains a list of directories to be searched whenever a command is typed, e.g. if PATH is set as follows:

```
X:\ADMINS\BIN;N:\ADMDIST\BIN
```

then when the user enters "admtrans" at the shell prompt the shell program will search for the executable file admtrans.exe first in the directory x:\admins\bin and then in the directory N:\admdist\bin.

Symbolic names on OpenVMS and the PATH on Win32 systems are examples of the "environment" that must be put in place in order to give users easy access to ADMINS tools and applications.

For OpenVMS this user environment consists of the symbolic names for commands (see Appendix C.5.1 “OpenVMS Symbols for ADMINS commands”) and logical names that are used to configure ADMINS (see Appendix B: “Special Logical Names used by ADMINS”), and also logical names and symbolic names that are used at the application level.

For Win32 this user environment consists of environment variables (including PATH) and logical names accessed via the ADMINS logical name server. The "process logical name table" and environmental variables can be used interchangeably[8] to configure the user environment for ADMINS and for ADMINS-based applications.

This user environment is usually put in place via command procedures that execute automatically at login (e.g. system-wide and user login command files on OpenVMS, and login scripts on Win32).

C.5.1 OpenVMS Symbols for ADMINS commands

ADMINS commands are referenced by OpenVMS symbols established when a user logs on to the system. The following list represents a standard list of these symbols. The examples in the Manual assume this symbol list. This list is also included on the ADMINS distribution as ADMSYMDEF.COM. The asterisk (*) in the symbol means that the symbol can be abbreviated at that point. Characters after the asterisk can be omitted. For example, entering any of the strings ACQ, ACQU, ACQUI, or ACQUIR will be interpreted as the command ACQUIR.

```
$ ACQ*UIR    :==  $ADM$DIST:ACQUIR
$ ADE*D           :==  $ADM$DIST:ADED
$ AD*IFF   :==  $ADM$DIST:ADIFF
$ AFU      :==  $ADM$DIST:AFU
$ AN*ALYZER       :==  $ADM$DIST:AN
$ AV              :==  $ADM$DIST:AV
$ BAC*KUP         :==  $ADM$DIST:AFU
$ CLR             :==  @ADM$DIST:CLR
$ CMP             :==  $ADM$DIST:CMP
$ COM             :==  @ADM$DIST:NATCOM
$ DAT*AP          :==  $ADM$DIST:DATAP
$ DEF*INE         :==  $ADM$DIST:DEFINE
$ ED              :==  $ADM$DIST:TED -CLM
$ EN*LARGE        :==  $ADM$DIST:ENLARG
$ FAC*QUIR        :==  $ADM$DIST:FACQUIR
$ FDA*TAP         :==  $ADM$DIST:FDATAP
$ FILECONVERT     :==  $ADM$DIST:FILECONVERT
$ FLAG*S          :==  $ADM$DIST:FLAGS
$ FL*OCK          :==  $ADM$DIST:FLOCK
$ IE              :==  $ADM$DIST:IE
$ MAI*NT          :==  $ADM$DIST:MAINT
$ MAN*UAL         :==  $ADM$DIST:MANUAL
$ MERGE           :==  $ADM$DIST:MERGE
$ ML*OCK          :==  $ADM$DIST:MLOCK
$ MOV*E           :==  $ADM$DIST:MOVE
$ MRG*FIL         :==  $ADM$DIST:MRGFIL
$ NATCOM          :==  $ADM$DIST:NATCOM
$ PASSW           :==  $ADM$DIST:PASSW
$ PREPROCESS      :==  $ADM$DIST:PREPROCESS
$ PRO*D           :==  $ADM$DIST:PROD
$ REP*ORT         :==  $ADM$DIST:REPORT
$ RNF             :==  $ADM$DIST:RNF
$ SCR*EEN         :==  $ADM$DIST:SCREEN
$ SEND            :==  $ADM$DIST:SEND
$ SOR*T           :==  $ADM$DIST:SORT
$ SEQ             :==  $ADM$DIST:FILECONVERT
$ SPR*OD          :==  $ADM$DIST:PROD
$ SYN*C           :==  $ADM$DIST:ADMSYNC
$ TAPCOPY         :==  $ADM$DIST:TAPCOPY
$ TAP*DMP         :==  $ADM$DIST:TAPDMP
```

8. Requests to translate logical names will return the value of an environment variable that matches the logical name if the logical name is not present in the process, desktop, or system table.

```
$ TAPSPL            :==  $ADM$DIST:TAPSPL
$ TED               :==  $ADM$DIST:TED
$ TPR               :==  $ADM$DIST:TPR
$ TRA*NS            :==  $ADM$DIST:TRANS
$ TXT*ACQ           :==  $ADM$DIST:TXTACQ
$ UDK                :==  $ADM$DIST:UDK
$ VIEW              :==  $ADM$DIST:TED -READ
```

C.5.2 Calling ADMINS Executables in the Windows Environment

For portability and compatability, ADMINS has developed techniques to allow developers and users to call ADMINS commands and some operating system utilities by the same names in the Win32 environment as they used[9] in OpenVMS.

ADMINS command files in the Win32 environment will utilize a "symbol definition" look-up table and substitute the "translation" (on the right of the table below) in the resultant Perl script for the "alias" (on the left of the table) found in the ADMINS command file. This table is a plain-text file enabled and identified by the logical name ADM_COMMANDS.

```
!================================================================
!   The logical name ADM_COMMANDS should contain the full path
!   name to this file when Adm2Perl is run.
!----------------------------------------------------------------
ade*d                   AdmAded
adef                    AdmDefine
adiff                   AdmDiff
afu                     AdmFU
av                      AdmAV
bac*kup                 AdmFU
cmp                     AdmCmp
cop*y                   AdmCpy
dat*e                   AdmDate
ddm                     AdmDDM
def*ine                 AdmDefine
del*ete                 AdmDel
diff                    AdmDiff
ed                      ted - clm
enl*arge                AdmEnlarge
exp*lode                AdmExplode
fac*quir                AdmFac
ie                      AdmIE
lcr                     AdmLcr
ldl                     AdmLdl
lsh                     AdmLsh
ltr                     AdmLtr
```

9. via ADMSYMDEF.COM as described in the previous section Appendix C.5.1 "OpenVMS Symbols for ADMINS commands".

```
mai*nt                  AdmMaint
mov*e                   AdmMove
mrg*fil                 AdmMrgFil
pas*sw                  AdmPassw
pro*d                   AdmProd
ren*ame                 AdmRen
rep*ort                 AdmReport
rnf                     AdmRnf
scr*een                 AdmScreen
sor*t                   AdmSort
tra*ns                  AdmTrans
```

To provide a similar capability in the "Command Prompt" or "Console" window, use a plain-text file containing DOSKEY macros, similar to the following file, named "doskey.macros":

```
add=AdmTrans ADM_DD_DIST:adm_dd_menu
ade=AdmAded $*
cmp=AdmCmp $*
com=AdmCom $*
cpy=AdmCpy $*
ddm=AdmDDM $*
def=AdmDefine $*
rem=AdmDel $*
fac=AdmFac $*
lcr=AdmLcr $*
ldl=AdmLdl $*
lsh=AdmLsh $*
ltr=AdmLtr $*
mai=AdmMaint $*
mov=AdmMove $*
pro=AdmProd $*
rep=AdmReport $*
scr=AdmScreen $*
sor=AdmSort $*
tra=AdmTrans $*
```

Enable these macros with the following command:

```
> doskey /macrofile=doskey.macros
```

Appendix D:Reserved Field Names

A number of "names" have specific meaning to certain ADMINS commands and the user should not use these "names" as field names. The following is a list of reserved "names" in ADMINS.

D.1 Reserved Field Name List

ADM$ENTER	The name of a special integer field that can instruct TRANS to act as if a field has been entered without actually entering any field (see Section 16.17 "ADM$ENTER: Force TRANS Field Entry Processing").
ADM$NLREC	The name of a field that allows the RMO to detect which record locks have been ignored (see Section 16.21.2 "ADM$NLREC: Identify Ignored Locks").
ADM$NOLOCK	The name of a field that allows the RMO to detect when the user elects to ignore a record lock at the "Wait or Ignore" prompt, or when file option "I" is in effect (see Section 16.21.1 "ADM$NOLOCK: Record Lock Ignored Flag")
ADM$SCRNAM	The name of a field that TRANS will automatically load with the current screen name (see Section 5.5.8.7 "ADM$SCRNAM").
ADM$SUBSCR	The name of a field that is used to track and change the active subscreen (see Section 16.16 "Subscreen Status and Control: ADM$SUBSCR").
ADM$RECNO	The name of a special integer field that can TRANS will set to the current record number in a multi-record screen (see Section 16.22.1 "ADM$RECNO: Record Position in Multi-Record Screen").
ADM$RECORDLOCK	MAINT sets this field and checks it to allow alternative ways of handling records that are locked (see Section 10.1.1 "ADM$RECORDLOCK")
ADM$TRONAM	The name of a field that TRANS will automatically load with the current TRO name (see Section 5.5.8.8 "ADM$TRONAM").
AND	The name "AND" is used as a logical operator for connecting comparison expressions to make Boolean expressions (see Section 8.4 "Logical Operators").

B$B	The name "B$B" is a local field in an RMO behind a screen which when set to a branch code causes an automatic branch in TRANS (see Section 16.2 "Automatic Branching: B$B and R$R").
B$OB	The name "B$OB" is a local field in an RMO behind a screen which is used to request a second end-of-record RMO call in TRANS (see Section 16.8 "Post-Writeback EOFREC RMO Call: B$OB").
B$fieldname	The name "B$fieldname" (where "fieldname" is any name) is used in TRANS as an actual or local field set to a value of a "screen-name" which can then be used as a branch location (see Section 5.7.3 "Calculated Branches").
B$KEYFIELDS	The name of a special TRANS field that enables a single calculated branch (see Section 5.7.3 "Calculated Branches") to be used with any set of key fields (see Section 16.20 "Calculated Branches with Variable Branch Keys").
BACKSPACE	The name "BACKSPACE" is a local field in a MAINT used to instruct MAINT to backspace any number of records (see Section 10.10 "Backspace Records: BACKSPACE").
BEGREC	The name "BEGREC" should not be used as a field name because of the communication between TRANS and the RMO behind the screen. When S$S is set to "BEGREC" it means the RMO call is a "beginning of record" call (see Section 15.1.1 "Status: S$S").
BET	The name "BET" is used as a comparison operator meaning "between" in an ADMINS expression (see Section 8.3 "Comparison and Special Operators").
C$C	The name "C$C" is a local field in an RMO behind a screen which is used to control the movement of the cursor in TRANS (see Section 16.3 "Cursor Control: C$C and C$MULREC").
CHGDAT	The name "CHGDAT" is used as a field name in an automatic field log meaning "change date" (see Section 6.5 "Field Logging"). Since KEY fields of the master file are also included as a field in the field log, "CHGDAT" should be avoided as a field name of a KEY field in a DEF.
CR	The name "CR" is used in a command file (see Chapter 14: "Command Files") to indicate the response to a command prompt is a "carriage return". Since field names are also used as responses to command prompts, "CR" should be avoided as a field name in a DEF.
D$D	The name "D$D" is a local field in an RMO used with PROD to control deletion of records in the lookup file (see Section 11.13 "Record Deletion: D$D"), or with MAINT to specify deletion of the current record (see Section 10.5 "Record Deletion: D$D").

D$IR	The name “D$IR” is the name of an internal field maintained by TRANS to provide the active screen with the user’s default directory (see Section 5.5.8.4 “D$IR: Default Directory”).
DI$DI	The name “DI$DI” is the name of an actual field in the detail field in a PROD which is used to control insertion of records into the lookup file (see Section 11.12 “Controlling Lookup File Insertion: DI$DI”).
DLC	The name "DLC" is the name of a field automatically added to a DEF when an automatic field log is created. The field "DLC" means "date of last change" (see Section 2.10 “Field Logs”) and should not be explicitly added to the DEF of a file which is to be logged unless the user intends the field to have the transaction sequence number function.
E$NDSCR	The name of a field used in TRANS to detect which manual (keystroke) method was used to attempt to exit the current screen (see Section 16.12 “Check Screen Exit Keystroke: E$NDSCR”)
E$RR	The name "E$RR" is the name of a field used in a TRANS LINK statement as the key to access a table of check statement error messages (see Section 5.5.6.1 “Table Driven Check Statement Error Messages”).
E$RRMSG	The name "E$RRMSG" is the name of a field used by TRANS to display values linked from a table of check statement error messages (see Section 5.5.6.1 “Table Driven Check Statement Error Messages”).
E$XIT	The name "E$XIT" is used as a local field in an RMO which is used to terminate the execution of a command file. E$XIT may be used in a MAINT (see Section 10.7 “Terminating a Command File: E$XIT”) or in an RMO used with PROD (see Section 11.14 “Terminating a Command File: E$XIT”).
ELSE	The name "ELSE" is part of the IF_THEN_ELSE_END structure used to conditionally compute a result (see Section 8.5 “Conditional Statements”).
END	The name "END" is part of the IF_THEN_ELSE_END structure used to conditionally compute a result (see Section 8.5 “Conditional Statements”).
EOF	The name "EOF" should not be used as a field name if the lookahead facility is used because of the special meaning of the NX$EOF field (see Section 16.9 “Look Ahead: NX$fieldname”).
EOFREC	The name "EOFREC" should not be used as a field name because of the communication between TRANS and the RMO behind the screen. When S$S is set to "EOFREC" it means the RMO call is an "end of record" call (see Section 15.1.1 “Status: S$S”).

EQ	The name "EQ" is used as a comparison operator meaning "equal to" in an ADMINS expression (see Section 8.3 “Comparison and Special Operators”).
F$F	The name "F$F" is a local field in an RMO behind a screen which is used to force the display of the first record in the master file in TRANS (see Section 16.7 “Top of File Control: F$F”).
FLDNAM	The name "FLDNAM" is used as a field name in an automatic field log meaning "field name of the changed field" (see Section 6.5 “Field Logging”). Since KEY fields of the master file are also included as a field in the field log, "FLDNAM" should be avoided as a field name of a KEY field in a DEF.
FLDTYP	The name "FLDTYP" is used as a field name in an automatic field log meaning "field type of the changed field" (see Section 6.5 “Field Logging”). Since KEY fields of the master file are also included as a field in the field log, "FLDTYP" should be avoided as a field name of a KEY field in a DEF.
F$UNCKEY	The name of a special local RMO field which causes TRANS to make a special RMO call whenever a function key is pressed by itself, or to "trap" what function key was used to terminate an entry into an editable field (see Section 16.15 “F$UNCKEY - Function Key Detection in RMO”).
G$+nnn	The name "G$+nnn" (where "nnn" is a number of words) is used in TRANS to skip over a section of the global area (see Section 5.5.9).
G$RP	The name "G$RP" is the name of an internal field maintained by TRANS to provide the active screen with the group number of the UIC under which the user is currently operating (see Section 5.5.8.5 “G$RP: UIC Group Number”).
G$TMO	The name "G$TMO" is a local field in an RMO behind a screen which is used to cause TRANS to "timeout" and exit (see Section 5.5.20 “TIMEOUT statement”).
G$fieldname	The name "G$fieldname" (where "fieldname" is any name except "TMO", "RP", and "+nnn" which have special meaning as described above) refers to a "global field" in TRANS. TRANS maps all field names that start with "G$" onto the global area (see Section 5.5.9 “Global Fields”).
GE	The name "GE" is used as a comparison operator meaning "greater than or equal to" in an ADMINS expression (see Section 8.3 “Comparison and Special Operators”).
GOSUB	The name "GOSUB" ("go to subroutine") enables an RMO to go to another paragraph, execute statements and then return to the statement following that GOSUB call (see Section 9.6.3 “The GOSUB Statement”).
GOTO	The name "GOTO" is used to transfer control to a labeled paragraph in an RMO (see Section 9.6.2 “Record Maintenance Statements”).

GT	The name "GT" is used as a comparison operator meaning "greater than" in an ADMINS expression (see Section 8.3 "Comparison and Special Operators").
H$CODE	The name "H$CODE" is a local array in an RMO behind a screen which is used to specify the highlighting codes for the fields specified in H$NAME in TRANS (see Section 16.5 "Highlighting Fields").
H$ELPNAME	The name "H$ELPNAME" is used to identify the section name to be displayed by the HELP in TRANS facility (see Section 6.14 "HELP in TRANS").
H$NAME	The name "H$NAME" is a local array in an RMO behind a screen which is used to specify fields which should have special highlighting effects in TRANS (see Section 16.5 "Highlighting Fields").
HEX	The name "HEX" should not be used as a field name in a file which is to be written to an external file via DATAP. In a TAP used with DATAP the field name HEX refers to a hexadecimal constant (see Section 17.4.2.3 "Literals and Hexadecimal Constants").
I$I	The name of a special PROD/NOMATCH local RMO field that enables the RMO to control insertion into the PROD lookup file (see Section 11.10 "NOMATCH qualifier: Functionality without LOOKUP link").
IF	The name "IF" is part of the IF_THEN_ELSE_END structure used to conditionally compute a result (see Section 8.5 "Conditional Statements").
INCL	The name "INCL" is used as a comparison operator meaning "includes" in an ADMINS expression (see Section 8.3 "Comparison and Special Operators").
LE	The name "LE" is used as a comparison operator meaning "less than or equal to" in an ADMINS expression (see Section 8.3 "Comparison and Special Operators").
LINE	The name "LINE" used as a field name in a file has a special meaning when the file is to be used as the output file of a TXTACQ (see Section 17.3 "TXTACQ: Acquire Text Files").
LINE2	The name "LINE2" used as a field name in a file has a special meaning when the file is to be used as the output file of a TXTACQ (see Section 17.3 "TXTACQ: Acquire Text Files").
LSEQ	The name "LSEQ" can be used as a field name in a field log meaning "last transaction sequence for the master file". If it is present it is used and updated when field logging occurs (see Section 6.5.3 "Expanded Field Log Facilities").
LT	The name "LT" is used as a comparison operator meaning "less than" in an ADMINS expression (see Section 8.3 "Comparison and Special Operators").

M$LOC	The name of a special TRANS local RMO field that is used to designate the location of the status line (see Section 16.11 “Status Line Control: M$MSG and M$LOC”).
M$M	The name "M$M" is a local field in an RMO behind a screen which contains the "mode" of each call of the RMO by TRANS (see Section 15.1.2 “Mode: M$M”).
M$MSG	The name of a special TRANS local RMO field that is used to set the contents of the status line (see Section 16.11 “Status Line Control: M$MSG and M$LOC”).
MODE	The name "MODE" can be used as a field name in a field log. If it is present it turns on record logging and it is updated when logging occurs (see Section 6.5.3 “Expanded Field Log Facilities”).
MULREC	The name "MULREC" should not be used as a field name because of the communication between TRANS and the RMO behind the screen. When S$S is set to "MULREC" it means the RMO call is a "multi-record" call on a multi-record screen (see Section 16.22 “Multi-Record RMO Support”).
NE	The name "NE" is used as a comparison operator meaning "not equal to" in an ADMINS expression (see Section 8.3 “Comparison and Special Operators”).
NEW	The name "NEW" is used as a field name in an automatic field log meaning "new field value" (see Section 6.5 “Field Logging”). Since KEY fields of the master file are also included as a field in the field log, "NEW" should be avoided as a field name of a KEY field in a DEF.
NOT	The name "NOT" is used as a logical operator for connecting comparison expressions to make Boolean expressions (see Section 8.4 “Logical Operators”).
NOW	The name "NOW" is an internal field in ADMINS which contains the current time in hours, minutes, and seconds. NOW may be used in a screen and in an RMO behind a screen (see Section 5.5.8.2 “NOW: Current Time”), in a REPORT (see Section 7.16 “Internal Field Names”), in a MAINT (see Section 10.9 “Internal Fields: TODAY, NOW, and TICKS”), or in an RMO used with PROD (see Section 11.10 “NOMATCH qualifier: Functionality without LOOKUP link”).
NX$EOF	The name "NX$EOF" is a local field used in conjunction with "NX$fieldname" fields (lookahead facility) and is set to "-1" when the current record is the last record in the file.
NX$fieldname	The name "NX$fieldname" (where "fieldname" is the name of a field in the master file) is a local field which is set to the value of the "fieldname" field in the record following the current record. This is the "lookahead" facility.

OLD	The name "OLD" is used as a field name in an automatic field log meaning "old field value" (see Section 6.5 "Field Logging"). Since KEY fields of the master file are also included as a field in the field log, "OLD" should be avoided as a field name of a KEY field in a DEF.
OPER	The name "OPER" can be used as a field name in a field log meaning "operator id". If it is present it is used and updated when field logging occurs (see Section 6.5.3 "Expanded Field Log Facilities").
OR	The name "OR" is used as a logical operator for connecting comparison expressions to make Boolean expressions (see Section 8.4 "Logical Operators").
OUTFILE	The name "OUTFILE" is used as a local field in a MAINT to specify the name of an output file (other than the master file) which is to be written (see Section 10.12 "Writing Other Files: OUTFILE/ OUTRECS").
OUTRECS	The name "OUTRECS" is used as a local field in a MAINT to specify the number of records which are to be written to the output specified in "OUTFILE" (see Section 10.12 "Writing Other Files: OUTFILE/ OUTRECS").
P$P	The name "P$P" is used as a local field in an RMO to print on-line messages. P$P may be used in a MAINT (see Section 10.8 "Printing On-line Messages: P$P") and in an RMO behind a screen (see Section 16.6 "Printing Messages: P$P").
PAGE	The name "PAGE" is an internal field in REPORT which holds the current page number and thus should not be used elsewhere in a REPORT.
PGBRK	The name "PGBRK" should not be used as a field name because of the communication between TRANS and the RMO behind the screen. When S$S is set to "PGBRK" it means the RMO call is a "page break" call on a multi-record screen (see Section 16.22 "Multi-Record RMO Support").
PGNO	The name "PGNO" is an internal field in REPORT which holds the current page number and can be used as a print field designator (see Section 7.16 "Internal Field Names").
PROD$LINK	The name of a special PROD/NOMATCH local RMO field that is used to identify whether or not the Detail file record linked to a record in the Lookup file (see Section 11.10 "NOMATCH qualifier: Functionality without LOOKUP link").
Q$Q	The name "Q$Q" is used as a local field in an RMO to stop the execution of the a MAINT (see Section 10.6 "Quitting Before End of File: Q$Q") or a PROD (see Section 11.15 "Quitting Before End of File: Q$Q") after processing the current record. In REPORT a created field named "Q$Q" is used to stop REPORT execution at the current record (see Section 7.13.9 "Quit Before the End of File: Q$Q").

R$R	The name "R$R" is a local field in an RMO behind a screen which when set causes an automatic return to the last branch in TRANS (see Section 16.2 "Automatic Branching: B$B and R$R").
RECPOS	The default name of an obsolete special purpose field that is automatically set by SORT to the relative record sequence number of the input record which created that output record. REPORT can then be instructed to use the contents of the RECPOS field in place of the key field in a LINK.
RET	The name "RET" (for 'return') enables an RMO to go to the statement just after the last executed GOSUB and continue processing from that point (see Section 9.6.3 "The GOSUB Statement").
REP$SECLEN	The name of a special REPORT local RMO field that is used to tell REPORT how many lines a DETAIL or SUMMARY section will contain before REPORT processes it (see Section 7.19.1 "REP$SECLEN - Controlling Section Length in the RMO").
RJ$RJ	The name "RJ$RJ" is a local field in an RMO behind a screen which is used to reject APPEND, INSERT, and DELETE operations in TRANS (see Section 16.1.2 "Reject APPEND, INSERT, UPDATE, DELETE, or Transfer").
RLKOUT	The name "RLKOUT" was required as a field name in a file that is be used with the pre-Version 3.0 ADMINS "record lockout" facility (obsolete - see Appendix O).
S$S	The name "S$S" is a local field in an RMO behind a screen which contains the "status" of each call of the RMO by TRANS (see Section 15.1.1 "Status: S$S").
S$SEL	The name "S$SEL" is a local field in an RMO behind a screen which is used to determine whether a record is to be displayed (i.e. selected) in TRANS (see Section 16.10 "Select Records: S$SEL").
SELECT	The name "SELECT" is a keyword in a DEF (see Section 2.5 "Record Selection") and also in a TAP (see Section 17.2.4 "TAP - SELECT Line") identifying the SELECT statement which is used to determine the records to be included in a MOVE, SORT, or ACQUIR.
SEQ	The name "SEQ" is used as a field name in an automatic field log meaning "sequence of a multi-line change" (see Section 6.5 "Field Logging"). Since KEY fields of the master file are also included as a field in the field log, "SEQ" should be avoided as a field name of a KEY field in a DEF.
SK$SK	The name "SK$SK" is a local field in an RMO behind a screen which is used to control the "skipping" of fields in TRANS (see Section 16.4 "Controlling the Skipping of Fields: SK$SK").
STOP	The name "STOP" is used to stop execution of an RMO on a particular record (see Section 9.6.2 "Record Maintenance Statements").

T$T	The name "T$T" is the name of an internal field maintained by TRANS to provide the active screen with the terminal number of the terminal using the screen (see Section 5.5.8.3 "Terminal Number").
THEN	The name "THEN" is part of the IF_THEN_ELSE_END structure used to conditionally compute a result (see Section 8.5 "Conditional Statements").
TICKS	The name "TICKS" is an internal field in ADMINS which contains the hundredth of a second of the current time. TICKS may be used in a MAINT (see Section 10.9 "Internal Fields: TODAY, NOW, and TICKS"), or in an RMO used with PROD (see Section 11.9 "Internal Fields: TODAY, NOW and TICKS").
TIME	The name "TIME" can be used as a field name in a field log meaning "time of logging". If it is present it is used and updated when field logging occurs (see Section 6.5.3 "Expanded Field Log Facilities").
TODAY	The name "TODAY" is an internal field in ADMINS which contains today's date. TODAY may be used in a screen and in an RMO behind a screen (see Section 5.5.8.1 "TODAY: Current Date"), in a REPORT (see Section 7.16 "Internal Field Names"), in a MAINT (see Section 10.9 "Internal Fields: TODAY, NOW, and TICKS"), or in an RMO used with PROD (see Section 11.9 "Internal Fields: TODAY, NOW and TICKS").
TSEQ	The name "TSEQ" is the name of a field automatically added to a DEF when an automatic field log is created. The field "TSEQ" means "transaction sequence number" (see Section 2.10 "Field Logs") and should not be explicitly added to the DEF of a file which is to be logged unless the user intends the field to have the transaction sequence number function.
TTNO	The name "TTNO" can be used as a field name in a field log meaning "terminal number". If it is present it is used and updated when field logging occurs (see Section 6.5.3 "Expanded Field Log Facilities").
TTYP	The name "TTYP" is used as a field name in an automatic field log meaning "transaction type" (see Section 6.5 "Field Logging"). Since KEY fields of the master file are also included as a field in the field log, "TTYP" should be avoided as a field name of a KEY field in a DEF.
TX$OPTION	*Sets various options for internal text editing. The following options are defined:* *1 - Run "Initfile" always. If the field already contains text, this text is replaced by the output from "Initfile."*
U$SER	The name "U$SER" is the name of an internal field maintained by TRANS to provide the active screen with the user number of the UIC under which the user is currently operating (see Section 5.5.8.6 "U$SER: UIC User Number").

W$W	The name "W$W" is used as a local field in an RMO to control the write back of records to the disk. W$W may be used in a MAINT (see Section 10.4 "Controlling Write Back: W$W"), in an RMO used with PROD (see Section 11.11 "Controlling Writeback and Output: W$W"), and in an RMO behind a screen (see Section 16.1 "Controlling Changes Written To Disk").
[PR}	The name "[PR]" is an internal field in REPORT which is a counter of the number of records printed this page and can be used as a print field designator (see Section 7.16 "Internal Field Names").
[TR]	The name "[TR]" is an internal field in REPORT which is a counter of the total number of records printed thus far and can be used as a print field designator (see Section 7.16 "Internal Field Names").

Appendix E:File Concepts

ADMINS data files contain fixed length records. A record's size is determined by the number of fields it contains, and the size of those fields. For example, a file with 10 A20 (20 character alphanumeric) fields will have a record size of 200 bytes, or 100 words, while a file with 10 I (2-byte integer) fields will have a record size of 20 bytes, or 10 words.

ADMINS locates records via the file's built-in index,[1] which relates the key value for each record to the record position for that record (see Appendix E.3 "Key Index Structure").

Variable length data is handled as small repeating groups in a master record, or as repeating records in a file where sequences of records have the same (partial) key. Linkage between related records is achieved in the application instruction files, not in the file definitions themselves.

ADMINS files are stored on disk in space pre-allocated by DEFINE. When DEFINE is used to create an ADMINS file for a particular number of records, DEFINE requests enough disk blocks to hold that number of records assuming that the built-in index area of the file will be "fully packed". A fully packed built-in index block is one where there is no more space available in the index block for additional record pointers. This assumption is valid when the file is built with ACQUIRE, MOVE, or SORT. These ADMINS commands build a file by **appending** records to the end of the file. The append operation always results in a fully packed index. Record insertion and deletion can lead to partially full index blocks.

E.1 Internal File Layout

The internal file structure changed fundamentally with the introduction of Level 3 files. The following two sections describe the differing internal layouts of Level2 files and Level 3 files.

1. Level 3 files can maintain secondary indexes automatically, as described in Section 2.7 "Multiple Indices". Secondary indexes can also be achieved at the application level (when, for example, Level 2 files are in use) by using other files as cross reference files pointing to records in the master file. Secondary index files are allocated by DEFINE, initially created using SORT, and then must be maintained by the INDEX feature of TRANS or by using SORT to recreate the index after the master file has been updated.

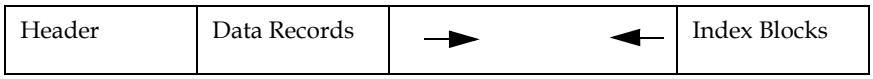
E.1.1 File Level 2

A keyed file contains three areas.

1. The **header area** which contains the information from the file "DEF" and file statistics. These statistics include the number of records in the file, the last record position used in the file, the last index block used in the file, and the location of the root index block.
2. The **records area** which contains fixed length records in the chronological order of how the records were added to the file. The last used record position kept in the header area marks the current boundary of the records area.
3. The built-in **key index area** which contains a hierarchical key indexing structure for the file in key value order. The key index area consists of index blocks, each equal in size to one ADMINS block,[2] extending from the **last** ADMINS block in the file up to the last index block used, as recorded in the file header. The area between the record position last used and the index block last used comprises the **available space** in the file. The indexing structure originates logically at the **root index block** whose address is recorded in the file header area.

The key values are stored **twice** in a file. Once in the record itself and again in the key index area as part of the pointer to the record. This double storage must be taken into account in estimating disk requirements for an installation.

The general layout of a Level2 ADMINS data file is:



A sequential file only has two areas, one for the header and one for the records. (The FILECONVERT task, described in Section 13.4.1 "Sequentialize an ADMINS data file", is used to bypass the key index structure by altering the header of the file to make a keyed file appear to be sequential.)

The File Header, containing meta-data information about the file, is in the front, and Data records are added immediately following the header, growing towards the end. Index blocks start at the end, growing forward. When the two meet, the file needs to be extended. That consists of adding a number of blocks to the end of the file, and moving the index blocks to the new end of the file. Except for the number of blocks in the file, and the movement of the index blocks to the new end of the file, nothing else changes.

E.1.2 File Level 3

The general layout of a Level 3 file is:

Header	Index Block(s)	Data Records	Index block(s)	Data Records	

Immediately following the File header is one or more index blocks, followed by data records and possibly new index blocks, intermixed and both growing towards the end of the file.

2. An ADMINS block contains 512 16 bit words (1024 bytes).

On Win32 Level 3 files have the immediate advantage that the file can grow dynamically even in a multi-user environment, thus eliminating multi-user exits because of lack of space. It also eliminates the need to get all multi-user users out of the file before it can be extended. →

Level 3 file structure enables the implemention of multiple indices, a feature that can be supported both on Win32 and OpenVMS. In addition, deleted records are marked as deleted, and do not reappear in the file if all indices are to be dropped and the indices rebuilt by reading the file sequentially.

E.2 File Operations

How are the basic file operations of record append, insert and delete performed?

Append	The record is added after the last record in the records area incrementing the last record position used and the key value is added after the last key index pointer in the key index area.[a]
Insert	The record is added just as in Append, only **the key index pointer is inserted in its proper sort order position** in the key index blocks of the file.
Delete	The key index pointer is deleted. The actual record is left in place.

a.Append mode is not available for alternate indices. When the initial record is input, the file is taken out of Append mode and put into Update mode automatically.

E.2.1 Finding Records by Key Value

Many critical operations in ADMINS applications, i.e. LINKs, PRODs (Lookup file), KEY statements KEY ranges, DELETEs, INSERTs etc. on are based on an optimized method for searching the key index for a particular key value. **This search method requires**[3] **that the keys in the index appear in sort order. (ADMINS cannot tell if the file is out of sort; any operation that depends on the key index search method assumes the index is in sort order). If the file is not in sort order, these operations should not be used.**

Other operations, e.g. simple MAINTs, MOVEs, REPORTs, PRODs (Detail file), etc., start at the beginning of the index and read the record referenced by each index pointer, until the last index pointer is encountered. These operations find all the records in the file even if it the index is not in sort.

3. When searching in an index block at any level of the hierarchy (see Appendix E.3 "Key Index Structure") if a key value is encountered that is higher than the target value, the method returns to the previous key value in the index block and uses pointer associated with that value to complete the search. This method cannot find all the records in the file if the key index is out of sort.

Appending records to a file is the most common way to put a sorted file out of sort order. For this reason **Append operations are not recommended.** Records should always be added to files by insertion. If appends are used, they should be used with caution. It is the responsibility of the application developer to make sure "out of sort" files are not accessed by key value.

E.3 Key Index Structure

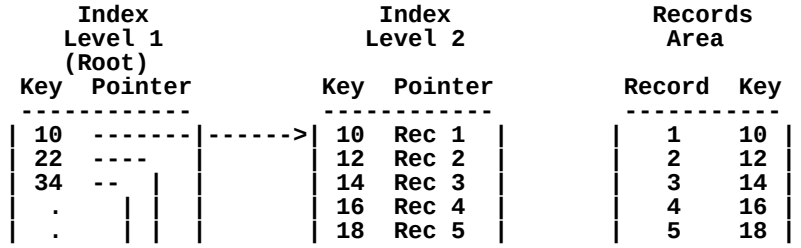
A key index is a hierarchical structure of sorted pointers. A pointer[4] either points to a lower level index block or to a record position in the file. A record search is performed starting at the **root index block** (identified in the file header) then proceeding down a level to another index block, until a pointer to a record position is reached or it is determined that the record is not in the file. ADMINS supports up to ten levels in the index hierarchy. This means that a record search can never perform more than ten disk seeks (nine for index blocks and one for the record; the root index block is kept in memory) in order to find a record.

Files very rarely have ten index levels unless they are huge to begin with and are then heavily updated via insert operations. In these cases the index should periodically be rebuilt to achieve full packing of the index pointers using MOVE or SORT. The typical file usually will have two, three, or at most four levels in the index structure and will therefore require two, three, or four disk seeks per record search.

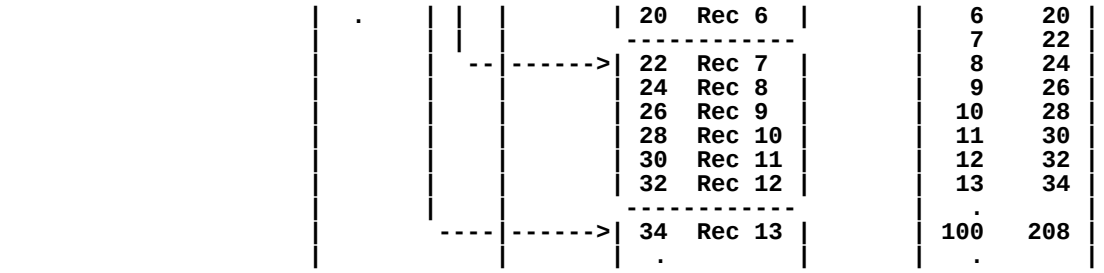
Index blocks may either be partially or completely full. Building a file via append operations packs the index blocks fully and therefore is usually the most efficient way to build a file. However, insert and delete operations may be required for file maintenance.

When ADMINS is asked to insert a record at a point in a file where the index blocks are full, ADMINS divides the full index block in half creating two half full index blocks, adds the record pointer to one of the half full index blocks, and then performs the record insertion. The pointer to the second index block (i.e. the new additional half full index block) is propagated back to the root index block. As required, full index blocks on this path back to the root index block are also split in half. This method of inserting adds levels to the index structure "very slowly". Only huge files have more than three or four levels.

For example, if we have the following structure.



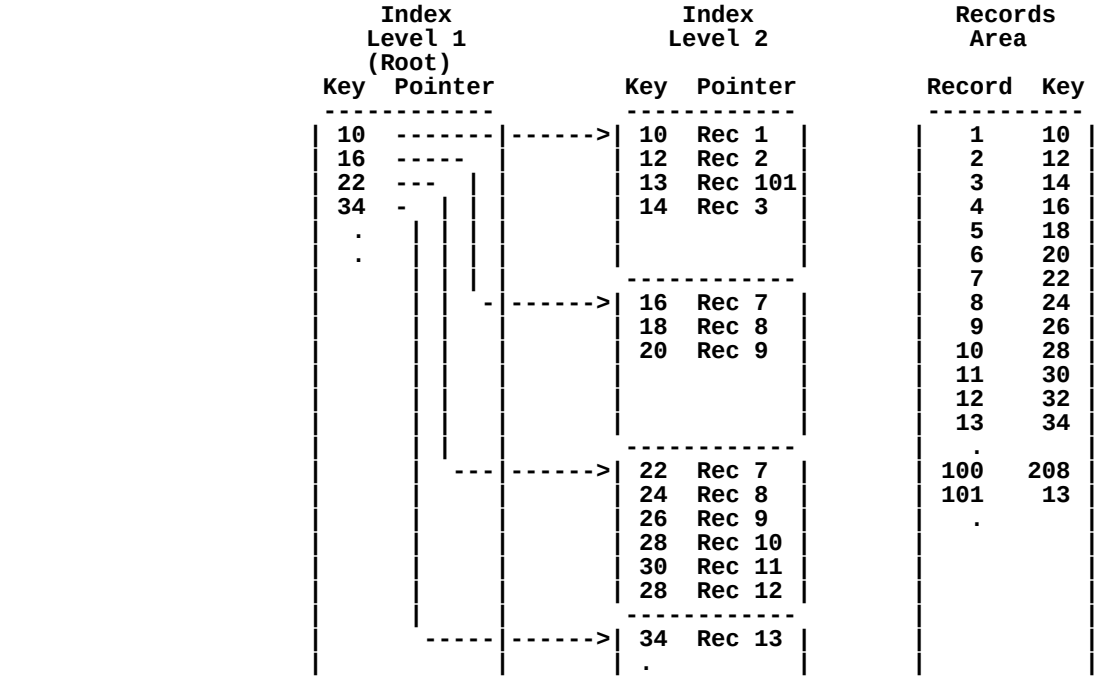
4. Files are created with 32-bit (4-byte) pointers, however if the file was created for less than 60,000 records by an older version of DEFINE or if "i" (lowercase) is in the string assigned to the logical name OPTION (see Appendix A: "Options") when a file is DEFINEd for less than 60,000 records, the file will have 16-bit (2-byte) pointers.



We see records with key values: 10, 12, 14, 16, 18, 20, 22, etc.

There are two levels of indexing: The root, level 2, and then the actual records area.

If we asked ADMINS to insert a record with a key value of "13", then the picture would look as follows:



As we see there are still two levels of indexing required to access records "14", "16", "18", and "20", but two of the index blocks are only partially filled. When the root block fills up, then we will require three levels of index blocks.

E.3.1 Structure Level

Three structure levels exist for ADMINS files. The oldest structure level, Level 0, is found in files created by versions of ADMINS prior to Version 3.2. Structure level 1 files are created by the commands of Versions 3.2, 3.3, and 3.4 of ADMINS. Structure level 2 files are created by Version 4.0 of ADMINS.

It is never necessary to re-define files with the older structure level (AFU "Describe" and "Detail Describe", described in Section 13.2 "AdmFu: ADMINS File Utility", give the structure level of the file). **All older files are fully forward compatible.**

And, for the most part, Level 1 files are backward compatible. When they are not, an appropriate error message is given.

The only Level 1 data files which are NOT compatible with earlier versions of ADMINS are:

1. Files with descending keys. (Descending keys are not supported in Level 0 files).
2. Files with at least one numeric key field (I, Dn, or Fn). (The FILECONVERT utility quickly converts such files back to the old format if necessary as described in see Section 13.4.2 "Convert Structure Level".)

Level 2 files are not backward compatible. You must convert Level 2 files (with FILECONVERT) to use them with earlier versions of ADMINS.

E.4 Available Space

The available space[5] in a file can be characterized in several ways.

1. The number of records that can still be appended to a file before the file becomes full, i.e. before the records area and the key index area overlap somewhere in the file. It is this number of available record spaces that the "AFU DESCRIBE" (Section 13.2.9 “Help”) and AV logical name A$RA (Section 13.6 “AdmAv - Communicate with ADMINS Files via Logical Names”) report as available space. This number is computed based on a record requiring the record size plus the key size plus the pointer size, and adding a factor for "breakage" (i.e. index blocks come in multiples of 512 16 bit words) and higher level key and pointer storage.
2. The number of records that still can be inserted at random points in the file. This number is harder to compute because it depends on the order and distribution of the insertions. However, given the "splitting index blocks" algorithm used for inserting new index entries, on average the index blocks are never more then half full. Hence if one allocates **twice** the numbers of index blocks than one would need if records were only being appended, and adds a factor (say five percent) to account for the higher level index blocks, there will always be enough space to store the inserted records.

5. Most ADMINS commands (except TRANS) can add records to a file will automatically enlarge the file if an impending overflow condition is detected (see Section 1.8 “Dynamic Data File Expansion”). If a file is opened multi-user however, it cannot be enlarged automatically.

Appendix F:Limits

This Appendix outlines the limits contained in the ADMINS commands.

F.1 DEFINE

The number of fields in a record cannot exceed 250.

The number of bytes per record cannot exceed 2,048.

The size of a single file is constrained only by the limits of ten index levels (see Appendix E: "File Concepts"), and by the limit that the total number of index blocks must be less than 2 raised to the 31st power (more than 2.1 billion!). That is, the size of a single file is unlimited for most practical uses.[1]

The number of characters in a field name can not exceed 18.

There may be up to 9 KEY fields and the combined number of characters of the sort key fields can not exceed 200.

The largest alphanumeric field is 80 characters (A80).

The number of characters in a picture field cannot exceed 18.

The range for an integer field (I) is +32,767 to -32,767.

The range of value that can be stored in a longword (32-bit) decimal field (L) is plus/minus 2,147,483,647 (2 to the 31st power minus 1). The maximum number of decimal places is 9.

The range of value that can be stored in a three word (48-bit) decimal field (D) is plus/minus 140,737,488,355,327 (2 to the 47th power minus 1). The maximum number of decimal places is 9.

The range of value that can be stored in a four word (64-bit) decimal field (F) is plus/minus 9,223,372,036,854,775,807 (2 to the 63rd power minus 1). The maximum number of decimal places is 9.

The highest date handled by the DA date format is 31DEC2060. The lowest date is 1-JAN-01. The DT date format handles dates in any year.

The SELECT statement in a DEF cannot exceed 180 characters.

1. It is theoretically possible to exceed ten index levels if the key size is extremely large (greater than 100 bytes), and large numbers of records are inserted non-randomly (i.e. at the same point in the file), without rebuilding the index via SORT.

F.2 SORT

The combined number of characters of the sort keys can not exceed 200.

There must also be adequate disk space for the input file, an output file of the same size as the input file, and a SORTXX.TMP file of the following size measured in 1,024 byte ADMINS disk blocks.

```
(((NWKEY + 1) / 2 + 2) * 2 * NRECS) / 500
```

Where:

NWKEY is the number of 16 bit words in the sort key.

NRECS is the number of records being sorted.

The three files involved in a sort can be on separate disks.

F.3 SCREEN

There may be up to 1000 editable fields referenced per screen.[2] In preparing multi-record screens one must multiply the number of editable fields displayed from one record times the number of records per screen to arrive at the number of editable fields on the screen.

There may be 60 open files per screen.

There may be 50 LINK paragraphs per screen. There may be 300 LINK fields per screen.

There may be up to 250 BOXes in a screen layout, resulting either from BOX statements or from boxes drawn[3] in the screen layout.

The Global Area is 1,024 words.

2. The limit on the number of fields that can be referenced in a screen (i.e. the screen's "virtual record") is 1,023 by default. This limit can be lowered by assigning a value in the range 250 to 1023 to the logical name ADM$MAX_FIELDS. Decreasing the limit frees up extra DA array space for other program functions.
3. see Section 5.5.10.1 “Drawing BOXes in the screen layout”

The number of branches allowed in a single TRS form is constrained by all of the following rules:

1. The number of words[4] required to store the names of all the screens in one TRS file should not exceed 90.
2. The number of words required to store all the **unique** branch descriptions (two lines of text) plus the unique branch codes should not exceed 2000.
3. The number of words required to store all the branch codes in a single BRANCHES paragraph plus the number of key fields in the BRANCHES paragraph plus an additional word per branch plus the text of the branch phrase should not exceed 2000.

The SCREEN command has an option which displays the capacity and utilization during compilation. To display this information, the SCREEN command is followed by the TRS name and the letter "C" on the command line.

```
$ screen test c
TEST.MAS: 234 WORDS IN DA
N200.MAS: 1885 WORDS IN DA
TEST READ
TEST:  61 DERIVED,  2119 FOR OPEN FILES, TOTAL DA: 2180
TEXT  ELES  LITS  BRTXT  BRWRDS  EDFLDS  VIREXT  NLINKS  EXTWRDS
USED  21    24    69    18     18      40      1        1       12
MAX   2000  2000  8000  2000   2000    1000    2000     100     4000
TEST: 89 OF 8500 WORDS IN TR
TEST COMPILED
1 SCREEN(S) COMPILED
```

Each of these statistics is described below. The numbers represent the number of words of storage.

TEXT	includes the messages in Check statements and Message statements, plus the source code of executable statements (Check, Message, Virtual). The limit is 2000 words.
ELES	is the space required to store the items in the FIELDS section. The limit is 2000 words.
LITS	is the storage for literal text in the SCREEN layout section. The limit is 8000 words.
BRTXT	is the storage for all the text in the BRANCHES paragraph. The limit is 2000 words.
BRWRDS	is the amount of object code which results from the BRANCHES paragraph. The limit is 2000 words.
EDFLDS	is the number of editable fields. The limit is 1000 editable fields.
VIREXT	is storage for LINK, APPEND, and INDEX fields names which are not in the DEF of the active file. For each such field SCREEN uses two 16-bit words, plus an additional word of storage for every two characters in the field name, rounded up (i.e. a five character field name counts the same as a six character field name). The limit is 2000 words.

4. When we refer to the storage of a text string in this appendix we are referring to the storage of the string in the "ADMINS string" format. The formula for computing storage size of an ADMINS string is (NC+3)/2 words, where NC is the number of characters in the text string.

NLINKS	is the number of LINK paragraphs. There may be up to 100 LINK paragraphs.
EXTWRDS	is the storage for object tables which result from external paragraphs, i.e. LINK, INDEX, and APPEND paragraphs. The limit is 4000 words.

The message in Pass 2 for each screen, "<n> OF 8500 WORDS IN TR", includes the branch tables, constants, object tables compiled from expressions, and check and message statement text.

F.4 TRANS

There may be up to 60 open files.

The "DA" array which holds TRANS' "virtual record" and the MD array which holds "metadata" (field names, types, "pics", etc.) contain 32,760 words each. The definition[5] of each open file plus the definitions of all local fields and arrays must fit in the MD array. The record buffer of each open file plus space for all local fields and arrays (field size times number of elements) must fit in the DA array.

All fields in external files count against these limits, even if they are not referenced in the screen or RMO. Total **referenced** fields, actual and virtual, are limited to 1,000 per screen.

Also note that there is always a global area (see Section 5.5.9 “Global Fields”) of 1,024 words in the DA array.

The General Editor Mode of TRANS is limited to the number of fields which can be displayed on the screen to a maximum of 1,000. It is variable depending on field types, field sizes, and whether 132 character screen width is used.

5. The size of the ADMINS data file’s self-contained definition can be obtained with AFU's DD option.

F.5 REPORT

REPORT can support up to 1000 fields, including actual fields in the file and derived fields. A derived field is created in one of the following ways:

1. Each CREATE statement creates one derived field.
2. Each field in a LINK or TABLE statement creates one derived field.
3. Each field operated upon by a TOTAL statement creates one derived field.

REPORT supports up to 10 levels of subtotaling. If SUPPRESS or RECODE is used, REPORT is limited to six levels of subtotaling.

REPORT has DA and MD arrays of 32,760 words and has limits similar to TRANS, i.e. up to 60 open files.

REPORT allows up to 1000 display fields, however, if automatic formatting is used REPORT is limited to 50 display fields.

REPORT supports up to 299 operations table elements. The operations that count against this limit are:

```
 1) each LINK statement
 2) each TABLE statement
 3) each EXECUTE statement
 4) each DIRECT statement
 5) each LAYOUT statement
 6) each SELECT or ORSELECT statement
 7) each RECODE statement
 8) each CREATE statement
 9) each TOTAL statement
10) each conditional SUPPRESS statement
11) the KEY statement (the KEY statement counts
    as two operations if it includes KEY$fieldname
     logical name references.
```

REPORT can print up to 63 lines in a single layout section (i.e. HEADING, DETAIL, PREVIEW or SUMMARY).

REPORT can handle up to 63 RECODE statements.

REPORT can output lines up to 254 characters wide (including carriage control characters in column one, if present).

F.6 CMP

The number of fields in a TABLE file cannot exceed 30.

There is a limit of 500 paragraphs in an RMS.

The total number of fields is 1000, including actual fields in the file, plus local fields and TABLE fields.

F.7 PROD

There can be up to 200 transfer fields in PROD.

There is no limit to the size of an RMO running with PROD.

PROD handles any valid ADMINS key, up to 100 words long, when linking to the lookup file.

F.8 ACQUIR

ACQUIR can read and process tapes with block sizes up to 40,000 bytes.

F.9 DATAP

DATAP can prepare output tapes with block sizes up to 40,000 bytes.

F.10 TAPDMP

TAPDMP can read and dump tapes with block sizes up to 40,000 bytes.

F.11 FACQUIR

FACQUIR can read and process files with record sizes up to 10,000 bytes.

F.12 FDATAP

FDATAP can prepare files with record sizes up to 2,048 bytes.

F.13 TAPCOPY

TAPCOPY can read tape record sizes up to 32,764 bytes.

Appendix G:

(This chapter's content only appears in the OpenVMS version of the Manual.)

Appendix H:Subroutines

This Appendix describes the subroutines that are in the ADMINS subroutine library. These subroutines may be used any place where an expression is permitted, namely:

1. Record maintenance procedures (RMO's) which can be used with TRANS, MAINT,MOVE, PROD, and REPORT.
2. SELECT, CREATE, and RECODE statements in a report instruction file (REP)
3. Virtual and Message fields and the Check statements in a screen instruction file (TRS)
4. SELECT statements in a file definition (DEF)

Generally, most subroutines are usable in most contexts, unless they are specifically designed for or limited to particular commands. This will be noted in the documentation.

H.1 Format of Presentation

For each subroutine there is a functional description, syntax presentation, and examples to illustrate the use if necessary. Field names used in the syntax as arguments are used for clarity. The same field names are not required in the actual use of the subroutine. Brackets, [ ], surrounding an argument in the syntax means that argument is optional.

H.2 Integer Decimal Values for ASCII Characters

Several subroutines refer to the integer decimal value of an ASCII character. The following is a table of all possible values:

Value	Character	Val	Char	Val	Char	Val	Char
0	ctrl/space	32	space	64	@	96	`
1	ctrl/A	33	!	65	A	97	a
2	ctrl/B	34	"	66	B	98	b
3	ctrl/C	35	#	67	C	99	c
4	ctrl/D	36	$	68	D	100	d
5	ctrl/E	37	%	69	E	101	e
6	ctrl/F	38	&	70	F	102	f
7	ctrl/G (bell)	39	'	71	G	103	g
8	ctrl/H (backspace)	40	(	72	H	104	h
9	ctrl/I (horizontal tab)	41	)	73	I	105	i
10	ctrl/J (linefeed)	42	*	74	J	106	j
11	ctrl/K (vertical tab)	43	+	75	K	107	k
12	ctrl/L (form-feed)	44	,	76	L	108	l
13	ctrl/M (return)	45	-	77	M	109	m
14	ctrl/N	46	.	78	N	110	n
15	ctrl/O	47	/	79	O	111	o

16	ctrl/P	48	0	80	P	112	p
17	ctrl/Q	49	1	81	Q	113	q
18	ctrl/R	50	2	82	R	114	r
19	ctrl/S	51	3	83	S	115	s
20	ctrl/T	52	4	84	T	116	t
21	ctrl/U	53	5	85	U	117	u
22	ctrl/V	54	6	86	V	118	v
23	ctrl/W	55	7	87	W	119	w
24	ctrl/X	56	8	88	X	120	x
25	ctrl/Y	57	9	89	Y	121	y
26	ctrl/Z	58	:	90	Z	122	z
27	ctrl/[(escape)	59	;	91	[	123	{
28	ctrl/\	60	<	92	\	124	|
29	ctrl/]	61	=	93	]	125	}
30	ctrl/~	62	>	94	^	126	~
31	ctrl/?	63	?	95	_	127	delete

H.3 Concatenation Subroutines

The concatenation subroutines, NCAT, FCAT, and CCAT are provided for composing fields of all types.

NCAT is designed to perform concatenation and field type conversion only.

FCAT is also designed to perform concatenation and field type conversion, but FCAT will interpret certain values in the arguments given to it as instructions for formatting the result field. FCAT can be used for retaining punctuation when converting numeric fields to alphanumeric fields, for right justification of data within a display field, or for placing custom-formatted date strings into a display field.

CCAT is the original ADMINS concatenation subroutine. It has been superseded by the two subroutines, NCAT and FCAT, to avoid a potential for ambiguity in interpretation of its arguments. In most cases CCAT will still function as it always has and need not be changed in existing procedures.[1] Either NCAT or FCAT should be used in new procedures.

1. CCAT originally could perform both simple concatenation and special formatting of the result fields. However, this dual capability created a potential ambiguity in interpretation of CCAT syntax that could cause unintended special formatting of the result field. If this ambiguous syntax is encountered by the compiler, the line number and the CCAT statement are printed under the message "The following CCAT is ambiguous. Use NCAT or FCAT." All such ambiguous CCATs are listed before the compiler exits. Each of these CCAT calls must be changed. If data reformatting is desired, change CCAT to FCAT (F stands for formatting). If data reformatting is not desired, change CCAT to NCAT (N stands for no formatting). Nothing else need be changed, just the name of the subroutine. The various uses of CCAT are for concatenating fields, converting fields between data types, and retaining punctuation when converting numeric to alphanumeric.

H.3.1 NCAT - Concatenating fields

NCAT concatenates the contents of one or more fields together and places the result in another field. (NCAT only operates on field names. A constant can always be placed in a field if a constant is needed as an argument for NCAT.) If all the fields were simply of type An, and the result field was large enough to hold the concatenated alphanumeric strings, then the operation of NCAT would be apparent. However, NCAT is **fully generalized** to operate on any number of arguments of any type.

How does NCAT work? First each argument is converted to a character string. The conversion rules for each data type is as follows:

1. Ln or Dn: Punctuation is removed and leading zeroes are inserted to create a 15 character field. For example, "12,376" becomes "000000000012376", "1.35" becomes "000000000000135", and "9,765,235.14" becomes "000000976523514".
2. I: Punctuation is removed, and leading zeroes are inserted to create a 5 character field. For example, "1" becomes "00001" and "1,279" becomes "01279".
3. DA or DT: Exactly as in a printout, e.g., "17-MAR-78", "January 11, 1997".
4. An: Trailing blanks are removed. A blank in an A1 field is not removed.
5. Xpic: Exactly as in a printout, e.g., "B000210".
6. Fn: Punctuation is removed and leading zeroes are inserted to create a 20 character field. For example, "12,376" becomes "00000000000000012376", "1.35" becomes "00000000000000000135", and "9,765,235.14" becomes "00000000000976523514".
7. TM: Exactly as in a printout, e.g., "11:44:35.07".

Then these converted strings are concatenated. The concatenated string is interpreted as an input string to the NCAT result field, and converted according to the type of that result field.

H.3.1.1 NCAT Syntax for Concatenating Fields

```
A = NCAT(A,B,C,D...)
```

A/__	Alphanumeric or picture field for the results of concatenation.
B/__	First field of the concatenation (any type).
C/__	Second field of the concatenation (any type).
D/__	Third field of the concatenation (any type). (Etc.)

H.3.1.2 NCAT Concatenation Examples

Given the following fields and values,

```
ACCT/XA999999
LET/A1 'B'
SEQ/I '21'
ZERO/X9 '0'
```

then

```
ACCT = NCAT(ACCT,LET,SEQ,ZERO)
```

would create an account number where ACCT would be "B000210".

Given the following fields and values,

```
NAME/A30
FNAME/A10 'JOHN'
INIT/A1 'A'
LNAME/A12 'SMITH'
BLANK/A1 ' '
```

then

```
NAME = NCAT(NAME,FNAME,BLANK,INIT,BLANK,LNAME)
```

would create a full NAME of "JOHN A SMITH".

H.3.2 NCAT - Converting Between Field Types

NCAT can be used to convert from any field type to another. NCAT recognizes when it is being used to convert one "numeric" type to another. A numeric type, i.e., Ln, Dn, I, DA, or Fn, is a type on which arithmetic can be performed. When NCAT is called with only two arguments which are both from those four types then NCAT does a direct conversion (in "binary") rather than converting via character strings. However, when NCAT is called with only two arguments, but one or both are of the type An or Xpic, then the conversion is done as described in Appendix H.3.1 "NCAT - Concatenating fields".

H.3.2.1 NCAT Syntax for Converting Fields

```
A = NCAT(A,B)
```

A/__	Dn, I, DA, Fn, An, Xpic type field for the result.
B/__	Dn, I, DA, Fn, An, Xpic type field to be converted. Remember, if both fields are of a numeric type, then the conversion is a binary conversion.

Converting a DA field to a D field works properly, that is a positive D field results even when the DA field is past 26-NOV-81, which treated as a 16-bit quantity is a negative integer.

H.3.2.2 NCAT Example of Converting Fields

For example, assume D is of type D, I of type I, DA of type DA and F of type F.

```
D = NCAT(D,DA)
I = NCAT(I,D)
DA = NCAT(DA,I)
D = NCAT(D,F)
```

All of the above are acceptable conversions. When converting to a smaller format, i.e. F to D, F to I, D to I, the most significant part of the number is lost. This is done because one is presumably converting to a "smaller" format because the most significant part of the "larger" format is not used. Also the conversion proceeds without regard to the number of decimal places in either field and does not automatically adjust the position of the decimal point. Decimal point alignment must be handled explicitly by the application developer.

H.3.3 FCAT - Retaining Punctuation

The FCAT subroutine may be instructed to retain all punctuation when concatenating numeric fields into alphanumeric fields. This is done by making the second argument to FCAT an alphanumeric A2 field containing the characters ".,". This instructs FCAT not to remove punctuation, nor to insert leading zeroes, in all numeric fields that follow in that call to FCAT. That is, when FCAT converts the D field to the 15 character field, the punctuation is left in.

FCAT also has the additional option to suppress the commas and leading zeroes when converting decimal to ASCII, but to leave the decimal point intact. This is a variation of the "." facility described above. This feature is requested by placing an A2 field containing "0," as the second argument to FCAT.

FCAT can also convert a D field to an A field and be instructed where to place the decimal point. This is done by letting the second argument be an A2 field containing ".D", and letting the third argument be an I field containing the number of decimal places.

H.3.3.1 FCAT Syntax for Retaining Punctuation

```
A = FCAT(A,DC,[NDEC,]B,C,D...)
```

A/__	Alphanumeric field for the result.	
DC/A2	'.,' or '0', or '.D'	
	'.,'	FCAT does not remove punctuation or insert leading zeroes in the numeric fields that follow.
	'0,'	FCAT suppresses the commas and leading zeroes when converting decimal fields to alpha, but leaves the decimal point intact.
	'.D'	FCAT places the decimal point in a position specified in the NDEC field when converting decimal to alpha.
NDEC/I	Number of decimal places when using '.D' in the DC field.	
B/__	First field of the concatenation (any type).	
C/__	Second field of the concatenation (any type).	
D/__	Third field of the concatenation (any type). (Etc.).	

H.3.3.2 FCAT Retaining Punctuation Examples

The feature for retaining punctuation is useful in formatting messages for the P$P field that contain numeric data. For example:

```
FILE ACCT.MAS
LOCAL
DC/A2  '.,'
P$P/A40
ACCOUNT/A7  'ACCOUNT'
MSG/A10  'BALANCE IS'
BLANK/A1  ' '
PROGRAM
P$P = FCAT(P$P,DC,ACCOUNT,BLANK,ACCT#,MSG,BALANCE)
```

The above example might print a message as follows:

```
ACCOUNT 34512 BALANCE IS 12,562.34
```

If the DC field had contained '0,' the message would have printed as follows:

```
ACCOUNT 34512 BALANCE IS 12562.34
```

The following is an example of decimal point placement.

```
...
LOCAL
DC/A2  '.D'
NDEC/I  2
PROGRAM
A12 = FCAT(A12,DC,NDEC,VALUE)
...
```

If VALUE was a "D" field containing the value "2,350", this will cause the A12 field to contain "23.50" after the FCAT is performed.

H.3.4 FCAT - Right Justify Decimal Values in Alpha Field

FCAT will right justify a decimal value concatenated into an alphanumeric field if an A2 field with value "RJ" precedes the decimal field in the subroutine argument. Alphanumeric fields are usually left justified.

```
A = FCAT(A,XX,D)
```

A/_	Alphanumeric field for the result of concatenation.
D/_	Ln, Dn, or Fn field.
XX/A2	'RJ'

H.3.4.1 FCAT Right Justification Example

Given the following fields and values,

```
DC/A2 '0,'
XX/A2 'RJ'
AL/A10
DEC/D2  1234.56
```

then

```
AL = FCAT(AL,DC,XX,DEC)
```

results in the value

```
'1234.56'
```

for the field AL. The leftmost three characters are blank in the alphanumeric string.

H.3.5 FCAT - Custom Formatted Dates in Alpha Fields

FCAT will place a custom-formatted conversion of a date (DA or DT) field into an alphanumeric field when an An field (or constant) that contains a date formatting string precedes the date field in the list of FCAT arguments. An FCAT date formatting string consists of two periods followed by a format specified in the same manner used with the logical name ADM$DATE (see Section 2.4.2 "Field Data Types").

For example, given the following fields and values:

```
ALPHA/A20
DATE/DA    '26-JAN-93'
FMTDAT/A10 '..m D, Y4'
```

Then:

```
ALPHA = FCAT(ALPHA,FMTDAT,DATE)
```

Would result in the string "January 26, 1993" being loaded into field ALPHA.

H.3.6 FCAT - Converting a String to a Date Field

FCAT can use a format argument to convert a string to a date field bypassing the current settings of ADM$DATE. The syntax is:

```
DATE = FCAT (DATE, FORMAT, FLD1 [,FLD2, FLD3...])
```

where DATE is a field type DA or DT, and FORMAT is an alpha field (or constant) which starts with two dots('..') followed by the date format.

The arguments that follow FORMAT can be a single alpha or picture (X...) field, or multiple fields which, when concatenated, create a date string in the form specified in FORMAT. For example:

```
LOCAL
STARTDATE/DA
DATEFMT/A10 '..Y4MD'
YEAR/X9999 '2005'
MONTH/X99  '05'
DAY/X99  '17'
PROGRAM
STARTDATE = FCAT (STARTDATE, DATEFMT, YEAR, MONTH, DAY)
```

results in the value "May 17, 2005" being loaded into the field STARTDATE regardless of the current ADM$DATE setting.

If the FORMAT contains all numerics three integer fields may be used to hold the year , month and day, e.g

```
DATE = FCAT(DATE,FORMAT,YEAR,MONTH,DAY
```

where FORMAT/A8 = '..Y4MD'.

H.3.7 FCAT - Format result with source field's edit mask

If FCAT's second argument contains ***'EM'*** FCAT will load the result field with the contents of the source field, formatted according to the source field's edit mask.

Example: If the ACCNT/A20 has an Edit Mask of '%E999%-%E999%-%E99999' the account displayed as "010-200-51000" is actually stored in the ACCNT field as "01020051000". Given a field WACCNT/A20 (without an Edit Mask) the following will be true:

```
WACCNT = ACCNT                ! WACCNT contains '01020051000'
WACCNT = NCAT(WACCNT,ACCNT)   ! WACCNT contains '01020051000'


FMT/A2 'EM'
WACCNT = FCAT(WACCNT,FMT,ACCNT)  ! WACCNT contains '010-200-51000'
```

H.4 Date and Time Subroutines

The subroutines in this group are all associated with either a date field (field type DA or DT) or a time field (field type A8, format HH:MM:SS or field type TM, format HH:MM:SS.TT).

H.4.1 TMDIFF - Difference Between Dates and Times

The TMDIFF subroutine is used to determine the time difference between two given points in time. TMDIFF accepts two pairs of date and time fields, or just two dates, or just two times, and returns the difference between in the specified format. TMDIFF can also be used to break a time field into its components, or to check the validity of time input in an A8 field.

TMDIFF Syntax:[2]

```
STAT = TMDIFF([DATE_1,DATE_2][,TIME_1,TIME_2][,D_DATE][,D_TIME])
```

STAT/I	Status Code
1	OK, and D_TIME, if any, is positive
-1	OK, and D_TIME is negative (needed when D_TIME is unsigned field type A8 or TM)
-2	Must have either 3 or 6 arguments
-3	D_DATE is not an L, D, or F field
-4	D_TIME is not an A8, TM, I, L, D, or F field
-5	DATE_n or TIME_n has bad field type
-6	A8 TIME_n field has invalid value
DATE_1/DA or DT	Field or constant containing starting date. If DATE_1 is given, DATE_2 and D_DATE must be supplied.
DATE_2/DA or DT	Field or constant containing ending date. If DATE_2 is given, DATE_1 and D_DATE must be supplied.
TIME_1/A8 or TM	Field or constant containing starting time. If TIME_1 is given, TIME_2 and D_TIME must be supplied. A8 time must be in 24-hour HH:MM:SS
TIME_2/A8 or TM	Field or constant containing ending time. If TIME_2 is given, TIME_1 and D_TIME must be supplied. A8 time is checked as for TIME_1.
D_DATE/ L,D,F	Field. If DATE_1 and DATE_2 are given, D_DATE is the positive or negative difference in days. No decimal places allowed in L, D, or F field.

2. Handling the numerous STAT return values is simpler than it may appear. STAT return values 1 and (-1) indicate a successful call: (1) means that D_TIME is positive; (-1) means that it is negative. All STAT error codes except (-6) result only from programming errors in the RMS, so, once the RMS is tested there is no need to write logic to handle them.

D_TIME/A8 or TM	Field. If TIME_1 and TIME_2 are given, D_TIME is the difference in A8 24-hour HH:MM:SS format or TM format. STAT tells whether D_TIM is positive or negative.
	or I(3) If D_TIME is an Integer array, the time difference is returned as three signed integers: D_TIME(1) is hours, D_TIME(2) is minutes; and D_TIME(3) is seconds. To ignore seconds, D_TIME can be an array of two integers rather than three; to ignore both minutes and seconds it can be a simple I field rather than an array.
	or L,D,F If D_TIME is a Long integer or Decimal field, the time difference is returned as a signed number of seconds. No decimal places are allowed in these fields.

Different time and date field types can be used freely in the same call to TMDIFF. If "ticks" (hundredths of a second) are required in time calculations **TM fields must be used for the TIME_1, TIME_2, and D_TIME arguments.**

Because TMDIFF can return the difference between two dates, with no times, or can return the difference between two times, with no dates, it has superseded the ADMINS subroutines DIFFDA and DIFFTM (see Appendix O: "Obsolete Commands and Syntax").

To break a time field (TM or A8) into its hour, minute, and second components, use TMDIFF to get the difference between 00:00:00 and the given time and place the result in an Integer array. For example, an RMS declares an integer array with three elements:

```
D_TIME/I(3)
```

and then calls TMDIFF as follows:

```
STAT = TMDIFF('00:00:00/TM',TIME_1,D_TIME)
```

To check the validity (format, etc.) of a time input into an A8 field,[3] Make a similar TMDIFF call as in the above example and check whether the STAT return is (-6).

Given the following fields and values,

```
STAT/I
DATE1/DA '17-NOV-92'
DATE2/DA '25-NOV-92'
TIME1/TM '13:35:00'
TIME2/TM '15:45:30'
DAYDIFF/L
TMARRAY/I(3)
```

then

```
STAT = TMDIFF(DATE1,DATE2,DAYDIFF)
```

would result in the value "8" in the field DAYDIFF, and

```
STAT = TMDIFF(DATE1,DATE2,TIME1,TIME2,DAYDIFF,TMARRAY)
```

would result in the value "8" in the field DAYDIFF, and the values 2 (hours), 10 (minutes), and 30 (seconds) being loaded into the three elements of the local integer array TMARRAY. (In each case STAT would be set to 1.)

3. Whenever possible TM fields should be used for time strings because they have automatic validity checking.

If the first two arguments are date (**DA** or **DT**) fields, and the third argument is an integer (I) array with at least three elements, **TMDIFF** returns the difference between the two date fields as years, months, and days in the integer array. For example:

```
BORN/DT   '23-Feb-1996'
DATE/DT   '30-Oct-1999'
AGE/I  (3)
STAT/I
...
STAT = TMDIFF(BORN,DATE,AGE)
```

would return **AGE**(1) = 3 (years). AGE(2) = 8 (months) and AGE(3) = 7 days (i.e. a person born on 23-Feb-1996 would be 3 years, 8 months, and 7 days old on 30-Oct-1999.

H.4.2 ADDA and ADDT - Add a Number of Days to a Date

ADDA and ADDT add or subtract a number of days to or from a date and place the result in second date field. Use ADDA to place the result in a field of type DA, or use ADDT to place the result in a field of the type DT.

H.4.2.1 ADDA and ADDT Syntax

```
DATE2 = ADDA(DATE1,NUMDAYS)
            or
DATE2 = ADDT(DATE1,NUMDAYS)
```

DATE2/DA or DT	Result of DATE1 + NUMDAYS. Field type will be DA if ADDA is called, DT if ADDT is called.
DATE1/DA or DT	Date to be added to.
NUMDAYS/I or L	Number of days to add. If the number is negative, then it is the number of days to subtract.

H.4.2.2 ADDA Example

Given the following fields and values,

```
I/I 8
DATE1/DA 17-NOV-79
DATE2/DA
```

then

```
DATE2 = ADDA(DATE1,I)
```

would result in the value "25-NOV-79" in the field DATE2.

H.4.3 ADDTM - Time and Date Calculation

The ADDTM subroutine adds a given number of seconds, minutes, hours, days, months, or years to a date or a date and time. ADDTM performs time and date arithmetic for a single time unit. For example, any number of months can be added to a date;[4] but a number of months and a number of days must be added separately, in two ADDTM calls.

When ADDTM is used to add or subtract seconds, minutes, or hours, the two last arguments (A8 time fields) are required. When the unit is days, months, or years, these arguments are not required.

ADDTM Syntax:

```
STAT = ADDTM(DATE_1,UNIT,DELTA,DATE_2,[TIME_1,TIME_2])
```

STAT/I	Status of the operation requested. 1 means the operation was successful. - 1 means there was an error in the arguments.
DATE_1/DA or DT	Initial date
UNIT/I	Time unit code, as follows: 2 seconds 3 minutes 4 hours 5 days 6 months 7 years
DELTA/I	The number of time units to add (positive or negative)
DATE_2/DA or DT	Date which results from the date or time calculation.
TIME_1/A8 or TM	Initial time, not required for date calculations. TIME_1 must have the format HH:MM:SS (leading zeroes are required).
TIME_2/A8 or TM	Time which results from a time calculation, not required for date calculations. TIME_2 has the format HH:MM:SS.

4. When some number of months are added to or subtracted from a date, the exact same day may not exist in the result month. In such cases (31-Mar plus one month, which means "the same day in April") ADDTM gives the last day of the result month as the result day (i.e.,30-Apr).

H.4.4 CHKDATE

The CHKDATE subroutine verifies that a date falls within a certain range of dates. The syntax is:

```
STAT = CHKDATE(DATE,BASEDATE,PRIOR,AFTER)
```

where:

STAT/I	1: OK, DATE is within range 0: DATE is outside range -1: DATE is not DA or DT field -2: BASEDATE is not DA or DT field
DATE	DA or DT field with date to check
BASEDATE	DA or DT field to check against
PRIOR/I	Number of days prior to BASEDATE that is allowed.
AFTER/I	Number of days after BASEDATE that is allowed.

Example:

```
IF CHKDATE(DATE,TODAY,30,5) EQ 1 THEN...
```

would check if DATE was within 30 days prior to TODAY, and 5 days later than TODAY.

H.4.5 Y$EAR - Extracting the Year from a Date

The Y$EAR subroutine is used to extract the year[5] from an ADMINS date field.

Syntax:

```
I = Y$EAR(DATE)


I/I         The year (4 digits)

DATE/DA     Date
or
DATE/DT
```

Given the following fields and values,

```
I/I
DATE/DA '15MAY2004'
```

then

```
I = Y$EAR(DATE)
```

would result in field I being set to the value "2004".

5. If "6" is included in the string assigned to the logical name OPTION (see Appendix A) then function of Y$EAR is modified so that it returns a value equal to the year of the date given less 1900, i.e. YEAR = Y$EAR(DATE) where DATE is set to July 23, 2004 will load the value "104" into YEAR. Ordinarily (without 6 in OPTION), the value "2004" would be loaded into YEAR.

H.4.6 M$ONTH Extracting the Month from a Date

The M$ONTH subroutine is used to extract the month as an integer (1 through 12) from an ADMINS date.

```
I = M$ONTH(DATE)

I/I          Result is an integer from 1 through
             12, where 1 = January, ... 12 = December.

DATE/DA      Date
or
DATE/DT
```

Given the following fields and values.

```
I/I
DATE/DA '15-MAY-84'
```

then

```
I = M$ONTH(DATE)
```

would result in the field I being set to the value "5".

H.4.7 D$AY Extracting the Day from a Date

The D$AY subroutine is used to extract the day of the month from an ADMINS date.

Syntax:

```
I = D$AY(DATE)

I/I          The result is an integer between 1
             and 31 containing the day of the month.

DATE/DA      Date
or
DATE/DT
```

Given the following fields and values,

```
I/I
DATE/DA 15-MAY-84
```

then

```
I = D$AY(DATE)
```

would result in the field I being set to the value "15".

H.4.8 TIMESTR - Extract Hours, Minutes, Seconds from Time

TIMESTR provides a flexible way to extract integer values for the hour, minute, and/or second from a field containing a time value. TIMESTR can also be used simply to check that an input time value is valid without returning any of its components. TIMESTR accepts a wide variety of time formats. The input time string can be stored in an alphanumeric (An) field, a time-of-day (TM) field, a picture (X) field such as X99A99 or X9999, or in an integer (I) field (hours and minutes only). Hours and minutes must always be present; and the components must be left to right order, hours first, then minutes, and then seconds (if present). Any single-character delimiter between components may be used; or the time string may have no delimiters. Minutes and seconds must be given in two digits; hours can be one or two digits. The following are examples of time formats that are acceptable to TIMESTR:

```
7:05      07:05    0705    07:05:06
7.05.06   070506   70506   705
```

TIMESTR is called using the following syntax:

```
STAT = TIMESTR(TIME,HOUR[,MIN[,SEC]])
```

TIME	Field containing time string in any of the formats described above. May be an An, X, TM, or I field.
HOUR/I	Field to receive value of hour. If not needed, a constant such as zero may be given as a place holder for HOUR.
MIN/I	Field to receive value of minute. If not needed, use a constant or, if SEC is not needed, omit MIN.
SEC/I	Field to receive value of second. If not needed, SEC may be omitted.
STAT/I	Status: 1 OK -1 Error in number of arguments or argument field types -2 Error in TIME string format or value

TIMESTR can return any, all, or none of the time string components. If the HR, MIN, and/or SEC argument is omitted, or if its position is occupied by a constant (i.e. zero), then the corresponding time component is not returned.

Some sample calls and results:

	Returns:

STAT = TIMESTR(TIME,HR,MIN,SEC)	Hours, minutes, seconds
STAT = TIMESTR(TIME,HR)	Hours
STAT = TIMESTR(TIME,HR,MIN)	Hours, minutes
STAT = TIMESTR(TIME,0,MIN)	Minutes
STAT = TIMESTR(TIME,0,0,SEC)	Seconds
STAT = TIMESTR(TIME,0)	(Format check only)

If the time string does not contain a value for seconds, seconds is returned as zero.

TIMESTR always makes the following checks on the time string (if the string fails one of these checks TIMESTR returns a STAT of -2):

1. There cannot be more than one non-numeric character in succession.
2. Hour and minute must be present.
3. Minute and second (if present) must be two characters long.
4. Hour cannot exceed 23; minute and second cannot exceed 59.
5. If hours, minutes, and seconds are all present, there must be either two delimiter characters which are the same, or no delimiter characters.
6. There cannot be any characters in the field after a valid time string.

H.5 Character String Handling Subroutines

ADMINS has a data type of alphanumeric string. The only operator for searching an alphanumeric field, INCL, is limited to searching for literal strings only. Although the character handling **operators** in ADMINS are limited, the subroutines described below do provide a complete character string handling facility in ADMINS.

H.5.1 STR - Select Part of a Field

The STR subroutine is provided to decompose fields of all types. STR will work on any type of field, but note that before the characters are selected, the field the data is to be taken from is converted to its output representation, left justified. Therefore, on An, Xpic, and DA fields, the characters selected will be where you expect them. However on I, Ln, Dn, and Fn fields, the output representation varies depending on the value of the number due to insertion of commas and decimal point. Before using STR on a numeric field, you should first convert the numeric field to an alphanumeric field using NCAT or FCAT (as described in Appendix H.3 "Concatenation Subroutines") and then use STR on the alphanumeric field.

H.5.1.1 STR Syntax

```
A = STR(A,B,J,K)
```

A/__	Alphanumeric or picture field for the result.
B/__	Field of any type that the data is to be taken from. Note however, that before the 'stringing' occurs, B is converted to its output representation form as in a printout.
J/I	Starting position in B (may be a constant as '3/I').
K/I	Ending position in B (may be a constant as '5/I').

H.5.1.2 STR Example

For example, assume an account number of the form X99999999 where the first two digits were the fund, the next three digits were the department, and the last three digits were the object. The following use of the STR subroutine and a check statement in a TRS would insure that only department "020" was entered.

```
...
E ACCT
V DEPT/X999 STR(DEPT,ACCT,'3/I','5/I')
C DEPT NE 020
DEPARTMENT 020 PLEASE
...
```

H.5.2 CASE - Convert Between Upper and Lower Case Letters

The CASE subroutine will change a character string from upper case to lower case or from lower case to upper case.

CASE will change the case starting at the character specified in LEFT up to the character specified in RIGHT depending on the value in WHICH.

H.5.2.1 CASE Syntax

```
NULL = CASE(FIELD,LEFT,RIGHT,WHICH)
```

NULL/I	Required for syntax purposes only.
FIELD/An	Field to be modified from LEFT through RIGHT.
LEFT/I	Starting position in FIELD.
RIGHT/I	Ending Position in FIELD.
WHICH/I	If 0: the conversion is to lower case. If 1: the conversion is to upper case. If 2: everything except the first character of each word is converted to lower case. For example: **JOHN N. JOHNSON** **will be converted to** **John N. Johnson**

H.5.2.2 CASE Example

Given the following fields and values,

```
NULL/I
NAME/A10 'Barbara'
LEFT/I 1
RIGHT/I 10
WHICH/I 1
```

then

```
NULL = CASE(NAME,LEFT,RIGHT,WHICH)
```

would result in an all upper case name, "BARBARA", in the field NAME.

H.5.3 SQUEEZ - Remove Extra Blanks in One or More Fields

The SQUEEZ routine can be used to remove extra blanks from an alphanumeric field. Consecutive alphanumeric fields **from the DEF of the file** (not local RMO fields) can be treated as one large alphanumeric field. After SQUEEZ is called the result will only contain single blanks between words and trailing blanks at the end of the field.

H.5.3.1 SQUEEZ Syntax

```
NULL = SQUEEZ(ALPH,NLETS)
```

NULL/I	Required for syntax purposes only.
ALPH/An	Field with which to begin removing extra blanks.
NLETS/I	Number indicating how many characters long ALPH is. NLETS may include multiple consecutive alphanumeric fields which follow ALPH. SQUEEZ is not sensitive to field boundaries, that is, fields are squeezed together as if they are one very large field.

H.5.3.2 SQUEEZ Example

Given the following file definition and field values in a file,

```
* TEST.DEF                Field Values
MAS 100                 ------------------------------
FIELD1 A30             "This  is an example of        "
FIELD2 A30             "how the  SQUEEZ  subroutine   "
FIELD3 A30             "works   on    multiple        "
FIELD4 A30             "fields.                       "
```

then executing the following RMO,

```
* TEST.RMS
*
FILE TEST.MAS
LOCAL
NULL/I
N/I 120
PROGRAM
NULL = SQUEEZ(FIELD1,N)
```

would result in the data being squeezed into FIELD1 through FIELD4 as follows:

```
FIELD1       "This is an example of how the "
FIELD2       "SQUEEZ subroutine works on mul"
FIELD3       "tiple fields.                 "
FIELD4       "                              "
```

H.5.4 SETRPL - Set Up Character Replacements for REPLAC

There are two subroutines to support character replacement. SETRPL is used once to activate a table of character translations ("replacements") that are used by REPLAC to convert characters in one or more alpha fields.

H.5.4.1 SETRPL Syntax

```
NULL = SETRPL(FROM,TO)
```

NULL/I	Required for syntax purposes only.
FROM/I(n)	Array containing integer decimal values of characters to be replaced. The last entry in the array is a -1 to indicate the end of the array.
TO/I(n)	Array the same size as the FROM array containing integer decimal values of the character to replace the corresponding character found in the FROM array. The last entry in the array is a -1 to indicate the end of the array.

H.5.5 REPLAC - Replace Characters Based on SETRPL

The second subroutine to support character replacement is called REPLAC. After SETRPL is used to activate a table of character translations then REPLAC is used to actually convert the characters in one or more alpha fields.

H.5.5.1 REPLAC Syntax

```
NULL = REPLAC(STRING,N)
```

NULL/I	Required for syntax purposes only.
STRING/An	Starting field on which the character replacement as established by the SETRPL subroutine is to be performed.
N/I	Number of consecutive An fields of the same length REPLAC subroutine is to be performed.

H.5.5.2 SETRPL and REPLAC Example

For example, consider the following DEF.

```
*  TRN.DEF
MAS 100
N I KEY1
A1 A30
A2 A30
A3 A30
```

The following RMO will run on TRN.MAS and convert the vowels in the fields A1, A2 and A3 to integers.

```
FILE TRN.MAS
LOCAL
FROM/I(7) 65 69 73 79 85 48 -1
TO/I(7)   49 50 51 52 53 32 -1
J/I 0
STAT/I
PROGRAM
IF J EQ 0 THEN J = 1 ; STAT = SETRPL(FROM,TO) END
STAT = REPLAC(A1,'3/I')
```

Note that the values for the ASCII characters A E I O and U are 65 69 73 79 85 and the values for the ASCII characters 1 2 3 4 5 are 49 50 51 52 53. (See Table in Appendix H.2 "Integer Decimal Values for ASCII Characters") The last entry in the FROM array is a -1 to indicate to SETRPL that it has reached the end of the table. The FROM and TO arrays also converts zero to blank (i.e. "48" to "32"). Note also that the SETRPL subroutine is only executed once and stays in effect for all subsequent executions of REPLAC.

H.5.6 LOCSTR - Locate a String Within a String

LOCSTR finds a string in another string. LOCSTR searches for the first (or last) occurrence of STR1 in STR2 starting at a specified character (FIRST) in STR2. If FIRST is negative LOCSTR returns the position of the last occurrence of the STR1 (the absolute value of FIRST identifies the starting position in STR2).

LENGTH is the length of STR1 **that is to be used** in the search. If LENGTH is negative LOCSTR performs a case insensitive search. (the absolute value of length indicates the length of STR1 that is to be used). POINT contains the integer pointer to the place in STR2 where STR1 was found. If POINT is returned as zero then STR1 could not be found in STR2.

H.5.6.1 LOCSTR Syntax

```
POINT = LOCSTR(STR1,STR2,FIRST,LENGTH)
```

POINT/I	Pointer to the place in STR2 where the first character of STR1 was found or zero if STR1 was not found.
STR1/An	Field containing the value that is to be located. This may be a constant.
STR2/An	Field that is to be searched.
FIRST/I	Starting point in STR2 where search is to begin. If FIRST is negative LOCSTR returns the position of the last occurrence of the string.
LENGTH/I	Length of STR1 that is to be used in the search. If LENGTH is negative LOCSTR will perform a case insensitive search.

H.5.6.2 LOCSTR Example

Given the following fields and values,

```
LOC/I
ARG/A1 ','
FIELD/A30 'Find a comma (,) in the text.'
START/I 1
LEN/I 1
```

then

```
LOC = LOCSTR(ARG,FIELD,START,LEN)
```

would result in the value "15" in the field "LOC".

H.5.7 LOCATE: Find a String within a String (any data type)

The LOCATE subroutine will search for the occurrence of a string of characters in another string. The main difference from LOCSTR is that the arguments may be any data type. They are converted to ASCII strings before the search starts. Thus it is possible to have an integer containing the value 123 and search for the occurrence of the string "123" within an alpha field. The syntax is:

```
POS = LOCATE(WHAT,STRING,START,FLAGS)
```

Where:

WHAT	Field containing the string to search for. May be any data type. Will be converted to ASCII before the search starts.
STRING	Field to search in. May be any data type. Will be converted to ASCII before search starts.
START/I	Position within STRING (after being converted to ASCII) where to start the search. May be field or constant. 0 (zero) is treated as 1 (one).
FLAGS/I	Bitwise mask to modify the behavior of the search. The following values are defined: 1: Reverse search (start from the back to find last occurrence). 2: Case blind search
POS/I	Returned with the position where WHAT was found in STRING, or 0 (zero) if no match was found.

The values in WHAT and STRING are converted to ASCII, left justified, and without punctuation or leading zeros.

H.5.8 CLEN - Find the Length of a String

CLEN returns the actual length of a character string in an alphanumeric field. That is, trailing blanks do **not** count as part of the string.

H.5.8.1 CLEN Syntax

```
LENGTH = CLEN(STRING)
```

STRING/An	Field to be used LENGTH/I. Actual length of the character string in STRING not including trailing blanks.

H.5.8.2 CLEN Example

Given the following fields and values,

```
LEN/I
FIELD/A60 'What is the length of this string?'
```

then

```
LEN = CLEN(FIELD)
```

would result in the value "34" in the field "LEN".

H.5.9 BLDSTR - Build a String From Another String

BLDSTR builds a new string from a section of an existing string. BLDSTR takes the FIRST through LAST characters of STR1 and places then in STR2 followed by trailing blanks.

H.5.9.1 BLDSTR Syntax

```
NULL = BLDSTR(STR1,FIRST,LAST,STR2)
```

NULL/I	Required for syntax purposes only.
STR1/An	Take the FIRST through the LAST characters of STR1 and place them in STR2 followed by trailing blanks.
FIRST/I	First character of STR1 to be used.
LAST/I	Last character of STR1 to be used.
STR2/An	Result field.

H.5.9.2 BLDSTR Example

Given the following fields and values,

```
NULL/I
STR1/A50 'Part of this field will be placed in STR2'
FIRST/I 9
LAST/I 23
STR2/A20
```

then

```
NULL = BLDSTR(STR1,FIRST,LAST,STR2)
```

would result in the field STR2 as follows:

```
STR2/A20 'this field will     '
```

H.5.10 INSTR - Insert a String into Another String

INSTR inserts a string into a section of another string. INSTR places the initial LENGTH characters of STR1 into STR2 starting at the FIRST character of STR2. That is, INSTR **overwrites** a section of STR2. Also to allow insertion of non-printable characters, e.g. escape sequences, an integer argument will be accepted in place of STR1. In this case LENGTH is assumed to be "1", regardless of its actual value.

H.5.10.1 INSTR Syntax

```
NULL = INSTR(STR1,STR2,FIRST,LENGTH)
```

NULL/I	Required for syntax purposes only.
STR1/An or I	Field to move from, or integer decimal value of a character.
STR2/An	Field to insert to.
FIRST/I	Starting point of the insert into STR2.
LENGTH/I	Number of characters of STR1 to use. Place the initial LENGTH characters of STR1 into STR2 starting at the FIRST character of STR2, overwriting a section of STR2. If STR1 is an integer decimal value of a character, the corresponding character is overwritten into STR2 (LENGTH is assumed to equal 1 regardless of actual value). This permits inserting non-printing characters such as ESCAPE into alpha fields.

INSTR supports spanning of multiple fields. If the LENGTH argument is negative, then the actual length of the STR2 argument is not checked. This allows INSTR to place values into any number of consecutively allocated An fields by referencing the first An field.

If the FIRST (starting point) argument is negative the actual length of the STR1 argument is not checked. This allows INSTR to read values from any number of consecutively allocated An fields by referencing the first An field.

Only consecutively defined An fields in a file DEF, G$ fields, and array elements are guaranteed to be consecutive.

NOTE

PLEASE NOTE: When using negative arguments with the INSTR subroutine you are bypassing the regular field length checking in ADMINS. It is the responsibility of the developer to insure that data is read and written correctly when ADMINS checking is bypassed.

H.5.10.2 INSTR Example

Given the following fields and values,

```
NULL/I
STR1/I 27
STR2/A4 ' (0l'
FIRST/I 1
LENGTH/I 1
```

then

```
NULL = INSTR(STR1,STR2,FIRST,LENGTH)
```

would result in the escape character (decimal 27) being placed in the first position of the field STR2. STR2 then would contain the string "<ESC>(0l" which is the escape sequence used to instruct a VT terminal to use the graphics character set.

H.5.11 OUTSTR - Extract a String from Another String

OUTSTR does the exact opposite of INSTR. OUTSTR can extract a string from a series of consecutively stored An fields.

A specified number of bytes from one string, starting at a specified place in that string, are moved into another string, using the following syntax:

```
NULL = OUTSTR(STR1,STR2,FIRST,LENGTH)
```

NULL/I	Required for syntax purposes only.
STR1/An	Field to be moved into.
STR2/An	Field to be moved from.
FIRST/I	Starting point in STR2 of the string to be moved into STR1.
LENGTH/I	Number of characters of STR2 to move.

If LENGTH is negative, OUTSTR does not check whether the FIRST position is within the actual length of STR2. Only as many characters as will fit in STR1 will be moved.

H.5.12 INTC - Find Integer Decimal Value of a Character

INTC returns the Nth character of STRING as an integer decimal value.

H.5.12.1 INTC Syntax

```
VALUE = INTC(STRING,N)
```

VALUE/I	Integer decimal value of the Nth character in STRING.
STRING/An	Field to use.
N/I	Character in STRING to convert to its integer decimal value.

H.5.12.2 INTC Example

Given the following fields and values,

```
VALUE/I
CHAR/A1 '^'
N/I 1
```

then

```
VALUE = INTC(CHAR,N)
```

would result in the value "94" in the field "VALUE". A table of the integer decimal value of all characters is included in Appendix H.2 "Integer Decimal Values for ASCII Characters".

H.5.13 FLDEQL - Find Value in Group of Fields

The FLDEQL subroutine is used to check if any of a number of consecutive, equal-length alpha fields has a given value. The syntax is:

```
FLDNO = FLDEQL(STR1,BASE,FIRST,LAST)
```

STR1/An	Alpha string to search for. STR1 can be a field or a constant.
BASE/An	The base field for numbering the fields to be searched. BASE must be a field.
FIRST/I	Field number, relative to BASE, where to start. BASE is field #1. FIRST can be a field or an integer constant.
LAST/I	Last field, relative to BASE, to use. The fields must all be alpha fields of the same length as the base field. Last can be a field or an integer constant.
FLDNO/I	0 if not found, else field # relative to BASE if STR1 matched that field.

H.5.13.1 FLDEQL Example

Assume the following .DEF:

```
MAS 100
IDENT   X9999  KEY1
NAME    A24
GRADE1  A4
GRADE2  A4
GRADE3  A4
GRADE4  A4
GRADE5  A4
GRADE6  A4
GRADE7  A4
GRADE8  A4
GRADE9  A4
GRADEA  A4
```

where the fields GRADE1 - GRADEA is used to store various degrees a person might have. A similar .DEF with the SELECT statement:

```
SELECT FLDEQL('MBA',GRADE1,1,10) GT 0
```

could be used to select all records that have the value 'MBA' in any of the 10 GRADEn fields.

Also, the following RMO statements could be used to detect if any record had more than one GRADEn field containing the value 'MBA':

```
LOCAL
FLDNO/I
START/I
.
PROGRAM
.
FLDNO = FLDEQL('MBA',GRADE1,1,10)
IF FLDNO GT 0 AND FLDNO LT 10 THEN ;
   START = FLDNO + 1 ;
   IF FLDEQL('MBA',GRADE1,START,10) GT 0 THEN ;
      ... more than one 'MBA' ...
```

H.5.14 FSEARCH - Find Character String in Group of Fields

The FSEARCH subroutine is used to search for a given text string anywhere within a number of consecutive alpha fields. The syntax is:

```
POS = FSEARCH(STR1,LEN,BASE,NFLDS,SPOS)
```

STR1/An	Alpha string to search for. STR1 can be a field or a constant.
LEN/I	Length of STR1 to use in the search. LEN can be a field or a constant.
BASE/An	The base field for the search. BASE must be a field.
NFLDS/I	Number of consecutive fields to use. NFLDS can be a field or a constant.
SPOS/I	Position number, relative to the beginning of BASE, where to start the search. SPOS = 1 will start the search from the beginning of BASE. SPOS can be a field or a constant.
POS/I	0 if not found, else position number relative to the beginning of BASE where the matching string was found.

H.5.14.1 FSEARCH Example

Assume the following .DEF:

```
MAS 1000
.
TXT1   A60
TXT2   A60
TXT3   A60
TXT4   A60
TXT5   A60
.
```

then the following RMO statement would locate the first occurrence of the string 'VAX 6240' anywhere within the TXT1 - TXT5 fields:

```
LOCAL
POS/I
FLDNO/I
SPOS/I
.
PROGRAM
.
POS = FSEARCH('VAX 6240',8,TXT1,5,1)
IF POS GT 0 THEN ;
   ... 'VAX 6240' was present ...
```

If you want to know which field it was found in,

```
FLDNO = (POS / 60) + 1
```

would give you the field number, relative to TXT1.

If you wanted to know if 'VAX 6240' was mentioned more than once, the following statement would give you the answer:

```
SPOS = POS + 8
IF SPOS LT 300 AND FSEARCH('VAX 6240',8,TXT1,5,SPOS) GT 0 THEN ;
         ... 'VAX 6240' present more than once ....
```

H.5.15 FORMAT - Format Alphanumeric Strings

The FORMAT subroutine provides a convenient way to format alphanumeric strings which contain values from various fields and/or literal text. Using FORMAT you can build strings up to 254 characters in length by placing the output in an array.

The FORMAT subroutine syntax is as follows:

```
STAT = FORMAT(FMTSTR [,FLD1,FLD2,...] ,RESULT)
```

FMTSTR/An	Format control string (field or alpha constant). Contains literal text and these special symbols: * represents value of next field (FLDn) AP) represents a literal apostrophe (') (SL) represents a literal slash ('/') (BL) represents a literal blank (' '), for inserting leading blanks in output string. (n) "tab to column n" place the next item at character position n in the output string (character position numbers start with zero, not one). Ignored if already past this position. N is between 1 and 254: that is, the control strings (1) through (254) are valid. =* represents a literal '*' character =(represents a literal '(' character == represents a literal '=' character
FLDn/any	Up to 14 fields of any data types whose values are substituted for each '*' symbol in FMTSTR.
RESULT/ An[(n)]	`Field (or local array) containing formatted result string`[a]
STAT/I	Status: 1 OK -1 FORMAT requires at least 2 arguments (FMTSTR and RESULT), and cannot have more than 16 arguments -2 FMTSTR and RESULT must be alphanumeric (An) -3 Internal format conversion error -4 Parenthesis error in FMTSTR: Unless preceded by '=', parentheses must contain either a column number or AP, SL, or BL. -5 Number of FLDn arguments is not same as number of '*' symbols in FMTSTR

a. FORMAT result arrays **must** be "local" arrays declared in the RMO. FORMAT treats the RESULT as a single "field" with a size equal to the size of the field times the number of elements in the array (maximum size 254 characters). For example, a RESULT A40 array with 6 elements is treated as a 240-character alpha field. If the string loaded into this array is 153 characters long, RESULT(1), RESULT(2), and RESULT(3) would have 40 characters loaded; RESULT(4) would have 33 characters loaded; and RESULT(5) and RESULT(6) would be blank.

In the ADMINS RMO syntax, an apostrophe is a constant delimiter and a slash delimits the value of a constant from its type. In ADMINS in general, leading blanks are squeezed out of alpha fields. Therefore, an attempt to place one of these characters directly in an alpha string usually fails. With the formatting symbols (AP), (SL), and (BL), FORMAT provides a straightforward way to place these troublesome characters in an output string. (AP) is converted to an apostrophe; (SL) is converted to a slash; and (BL) can be used to begin the output string with a blank. Note that regardless how many leading blanks are desired in the output string, you need to use (BL) only once, at the beginning of the format string.

When formatting numeric fields (I, Ln, Dn, and Fn), FORMAT never inserts leading zeros; always inserts commas (or dots if K is in the string assigned to the logical name OPTION (see Appendix A)); and never discards the decimal point indicator.

Some advantages of FORMAT over other methods are:

1. To simply concatenate fields separated by blanks, you do not need any local fields containing blanks if you use FORMAT. If you use NCAT or FCAT, you need them.
2. FORMAT provides an easy solution to the problem of inserting apostrophes, slashes, and leading blanks in alpha fields. This can be done with INSTR; but FORMAT is more straightforward.
3. In general, FORMAT has greater capacity than NCAT and FCAT, because NCAT and FCAT usually require more arguments to do the same thing. An RMO subroutine call can never have more than 16 arguments; and this limit is more likely to be reached with NCAT and FCAT than with FORMAT.

H.5.15.1 FORMAT Example

The following example illustrates the use of the FORMAT subroutine.

```
    DATE/DA                  ! Build string with FORMAT
    NAME/A30                 ! ------------------------
    AMOUNT/D2
    RESULT/A60
    STAT/I
    PROGRAM
    STAT = FORMAT('As of *, I owed * $*.
A24',DATE,NAME,AMOUNT,RESULT)
```

FORMAT substitutes the values of the fields DATE, NAME, and AMOUNT, in order, where the '*' symbols occur in the format string. If the values of the fields DATE, NAME, and AMOUNT are respectively '03-AUG-89', 'Kevin', and '1.00', then, after the FORMAT call, the RESULT field contains:

```
    As of 03-AUG-89, I owed Kevin $1.00.
```

H.5.16 STRTYP - Check Format of Alphanumeric String

The STRTYP subroutine checks the value of an alphanumeric (An) field to see if it can be successfully converted into a field of some other data type. STRTYP performs the same checks as the automatic data entry format check in TRANS.

STRTYP is useful for checking input when different kinds of data can be entered in a single alphanumeric field, and in other cases where alphanumeric data will later be converted to another data type.

The syntax of the STRTYP subroutine is as follows:

```
STAT = STRTYP(STRING,TYPE)
```

STRING/An	Field containing alphanumeric data to check.
TYPE/An	Field or constant containing a data type (I, D2, X999, etc.).
STAT/I	Status: 1 OK: STRING matches TYPE 0: STRING does not match TYPE -1: TYPE is not valid

H.5.16.1 STRTYP Example

STRTYP allows an application to be designed to accept data into an alpha field, while ensuring that the data is in the proper format to be subsequently converted to another field type. In the following example, the ASKSCR (see Appendix H.14.1 "ASKSCR: Prompt directly from RMO") subroutine prompts for a string in date format, and STRTYP is used to ensure the response is correctly formatted:

```
LOCAL
ASTAT/I
SSTAT/I
ESTAT/I
PROMPT/A34 'Enter the starting date DD-MMM-YY:'
EPROMPT/A24
Y/I 10
X/I 10
Z/I 11
ANSWER/A9
TYPE/A2 'DA'
PROGRAM
* Prompt for start date, and check format of reply
STARTPROMPT: ASTAT = ASKSCR(Y,X,PROMPT,ANSWER)
EPROMPT = 'Format error, try again' ;
SSTAT = STRTYP(ANSWER,TYPE) ;
 IF SSTAT NE 1 THEN ESTAT = ASKSCR(Z,X,EPROMPT) ;
      GOTO STARTPROMPT ;
  ELSE EPROMPT = ' ' ; ESTAT = ASKSCR(Z,X,EPROMPT) ; END
```

H.5.17 CHECKCHAR - validate field contents for special purpose

The CHECKCHAR subroutine is used to check that an alpha (An) field contains only characters from a certain subset of characters. For example, this subroutine provides an easy way to ensure that a field contains no "illegal" characters for use in an email address or web site name. The syntax is:

```
STAT = CHECKCHAR(MASK,STRING)
```

where:

STAT/I		Return Values
	1	OK, only valid characters
	0	Illegal characters presen
	-1	Non-printing character present (ASCII codes 1-31)
MASK/I		A code to determine which characters are legal:
	1	Uppercase 7-bit alphabetical (A-Z)
	2	Lowercase 7-bit alphabetical (a-z)
	4	Numerical (0-9)
	8	_ (Underscore)
	16	- (Hyphen)
	32	Space
	64	Any 7-bit punctuation character
	128	Any uppercase character (7 and 8 bit)
	256	Any lowercase character (7 and 8 bit)
	512	Any punctuation character (7 and 8 bit)
	1024	URL
	2048	Email address
STRING/An		An alpha string to check

Example:

```
IF CHECKCHAR(11,TEXT) LE 0 THEN …
```

Would check if TEXT contains only 7 bit alphabetical characters (both upper and lower case) and underscores (1 + 2 + 8 = 11).

The CHECKCHAR subroutine also checks that an A field contains no non-printable characters (control characters in the range 0 - 31). If such a character is found a -1 is returned. **This check is performed first** regardless of what character combinations CHECKCHAR is being asked to check for.

H.5.18 SPLIT - Splitting an Alpha String into Several Fields

The SPLIT subroutine splits an alpha string into several fields.

The general syntax is:

```
STAT = SPLIT(INPUT,SEP,OUT1 [,OUT2 ...])
```

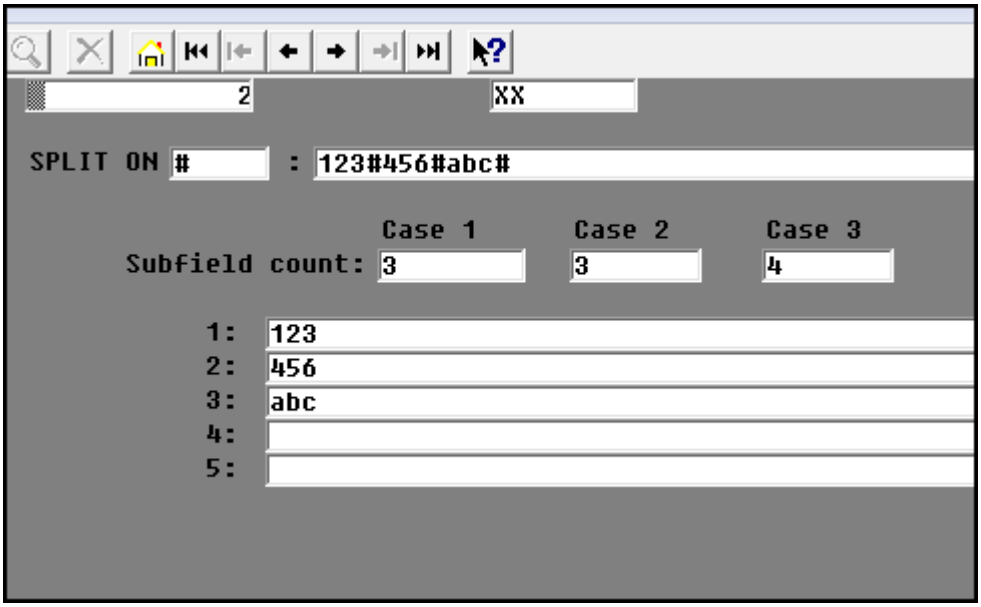
where:

INPUT/An	An An field containing the string to be split into separate fields. Each sub-field is separated by the SEP character(s).
SEP	Character(s) that separates the sub-fields in INPUT. SEP may be an An field, in which case it contains up to eight characters that serve as separators between the sub-fields, or an I field, in which case it contains the decimal value of a single character that serves as separator (e.g. SEP/I = 9 to indicate that the TAB character is used as separator). Positive integer values of SEP are normally used only if the separator character is non-printable. If the SEP argument is an integer and is negative, then the behavior of the subroutine is changed. When the last nonblank character in INPUT is the SEP character, SPLIT will load ran additional blank subfield, and report an additional subfield in STAT. See the example below.
OUTn	One or more fields to receive the values of the sub-fields. The values of the sub-fields specified in the INPUT field must have a data format that match the data type of the corresponding OUTn field. Instead of listing each output field as separate arguments, the output field can be an array. If the output array is of data type An, where n is between 6 and 18, e.g. OUT/A18(10), and the first element in the output array (e.g. OUT(1)) contains the value '@@', the rest of the array contains a list of field names to receive the output values. In all other cases the elements of the output array will receive the output values, i.e. the first value in OUT(1), the second in OUT(2) etc. If an array is used, it must be the ONLY output field.
STAT/I	Return status. > 0: Number of sub-fields found 0: No sub-field values found -n (0 < n < 99): Argument n has invalid format -99: The SEP field has invalid type (must be An or I) -98: Separator of type I must be <=255 -100: Too few elements in the output array -10n: Field name in array(n) not found

In the followiing example RMS the input string is split using the three alternatives for SEP, an ASCII character, an integer, and a negative integer,

```
file testdir:n.mas
m$m/a2
s$s/a10
longstr/a60
1sub/a40
2sub/a40
3sub/a40
4sub/a40
5sub/a40
*
asep/a1 '#'
*
isep/i 35      !ASCII code for '#'
*
negsep/i -35   ! negative integer signals alternate behavior
*              ! when final input character is a separator
1splcnt/i
2splcnt/i
3splcnt/i
program
if m$m ne 'UP' then STOP ; END
if s$s ne 'LONGSTR' then STOP ; END
1sub = '' ; 2sub = '' ; 3sub = '' ; 4sub = '' ; 5sub = '' ;
1splcnt = split(LONGSTR,ASEP,1SUB,2SUB,3SUB,4SUB,5SUB)  !Case 1
1sub = '' ; 2sub = '' ; 3sub = '' ; 4sub = '' ; 5sub = '' ;
2splcnt = split(LONGSTR,ISEP,1SUB,2SUB,3SUB,4SUB,5SUB)  !Case 2
1sub = '' ; 2sub = '' ; 3sub = '' ; 4sub = '' ; 5sub = '' ;
3splcnt = split(LONGSTR,NEGSEP,1SUB,2SUB,3SUB,4SUB,5SUB)  !Case 3
```

The screen shot below shows the results, note that the negative integer argument results in an extra reported subfield because the last chracter in the input is a seperator.



H.5.19 QUOTE - Place quotes around a character string

The purpose of the QUOTE subroutine is to put quotes around a character string, e.g. a file name with space(s) in the pathname. The syntax is:

```
STAT = QUOTE(FNM)
STAT = QUOTE(FNM,WHICH)
```

where:

STAT/I	For syntax only.
FNM/An	FNM is an alpha field (FNM/An) and can be a single field or an array (FNM/An(dim)).
WHICH/I	(Optional, field or constant) If called with only one argument the "" (double-quote) character will be inserted at the beginning and end of the alpha field. If present, allowable values are: 1 = Use single quotes (') 2 = Use double quotes (")

H.5.19.1 QUOTE Example

The requirement is to spawn NOTEPAD to display a text file (from an RMO). The path to the text file may contain an embedded blank, so if you tried to spawn e.g.

```
local
...
filename/a60 'c:\myfiles\latest message.txt'
command/a80
...
program
...
stat = format('notepad */a80',filename,command)
stat = spawn(command)
```

the spawned command would be:

```
notepad c:\myfiles\latest message.txt
```

which would likely produce an error message (from notepad) indicating

```
c:\myfiles\latest not found
```

To prevent this, use QUOTE to place quotes around the file argument, so the command line will not be misinterpreted

```
local
...
filename/a60 'c:\myfiles\latest message.txt'
command/a80
...
program
...
stat = quote(filename)
stat = format('notepad */a80',filename,command)
stat = spawn(command)
```

which would spawn the command

```
notepad "c:\myfiles\latest message.txt"
```

H.6 Text Handling Subroutines

The TEXTCOPY subroutine is used for moving data between two TXnn or TInn fields, or for moving data between alphanumeric fields or arrays and a TXnn or TInn field. The TEXTATTR subroutine provides access to the attributes of the text stored in a TXnn or TInn field. The SEARCH subroutine searches for a specified string in a TXnn or TInn field.

Two other subroutines are provided that allow: consecutive alphanumeric fields of the same size in the main file of a screen to be edited as a paragraph (the EDIT subroutine); a generalized capability to justify several consecutive alphanumeric fields as a paragraph (the PARAG subroutine).

H.6.1 TEXTCOPY: Move Information Between Text Fields

Use the TEXTCOPY subroutine to copy text in one text field or array of alphanumeric fields to another text field or array of alphanumeric fields.

Syntax:

```
STAT = TEXTCOPY(FROMFLD,TOFLD,OPTION[,DIM[,TOPRUL[,BOTRUL]]])
```

STAT/I (Value Returned)		Status:
	1:	OK: Text was successfully copied
	0:	FROMFLD was empty, no text copied
	-1:	FROMFLD has improper data type
	-2:	TOFLD has improper data type
	-3:	OPTION has improper data type
	-4:	OPTION has invalid value
	-5:	Output file is open Read Only
	-6:	TOFLD does not provide File Name
	-7:	Unable to create output file
	-8:	Too few arguments (must be at least 3)
	-9:	Too many arguments
	-10:	DIM is not of type integer
	-11:	TOPRUL is not of type integer
	-12:	TOPRUL not found in Data Dictionary
	-13:	BOTRUL is not of type integer
	-14:	BOTRUL not found in Data Dictionary
	-15:	Invalid From field/Option combination
	-16:	TO field invalid text format
	-17:	Unable to open work file for RTF format
	-18:	Unknown error getting internal RTF text
	-19:	Error removing old lock mark
	-20:	Cannot open input text file
	-21:	Cannot create the output file
FROMFLD/TIn Txn An		Text field to copy from. This field can be a text field (internal or external), or an array of alphanumeric fields (e.g. ALPHA/A60 (6)).

TOFLD/Tin TXn An(n)		Text field to copy to. If FROMFLD is internal text and TOFLD is external text, and no text is presently in TOFLD or you are replacing the current text in the external TOFLD, a file name must be provided in this field. If TOFLD is alphanumeric, DIM lines of text will be loaded into the TOFLD array. TEXTCOPY may be used to copy data from one array of alphanumeric fields to another array of alphanumeric fields, perhaps with a different field length.
OPTION/I		`Copy option`
	0	Replace text in TOFLD with text in FROMFLD
	1	Append text in FROMFLD to text in TOFLD
	2	Insert text in FROMFLD at top of text in TOFLD
	3	If FROMFLD is text and TOFLD is alphanumeric TOPRUL contains line number to start at in FROMFLD, load DIM lines in TOFLD array. If FROMFLD is alphanumeric and TOFLD is internal text (TI), replace DIM lines in TOFLD starting at line TOPRUL.
	5	Append text in FROMFLD to text in TOFLD, but use existing ruler in TOFLD rather than ruler in FROMFLD.
	8	FROM is an alpha field containing the file name of a text file to copy to the TO field. The text must be plain text.
	16	TO field is an alpha field containing the file name where the output text is sent as a plain ASCII file.
	32	TO field is an alpha field containing the file name where the output text is sent in RTF format.
	64	From field is an An field containing the name of an RTF file to be copied/appended/inserted into the To field
	+1000	If you add 1000 to the OPTION setting, any locked line mark in the from text will be removed when copied to the To text field.
	+2000	If you add 2000 to the OPTION setting, control characters (bolding, underline, etc.) will be removed when copying from a text field to an array of alphanumeric fields.
	+4000	Act as if a hard carriage return occurs at the end of each field when copying from an array of alpha fields (i.e. result will not be normalized
	+8000	Insert a PageBreak after the text being inserted at the top, or before the text being appended at the end.
	+16000	If a text field is defined as using AdmTED as its editor, override this to use WinWord when creating a new piece of text..

DIM/I	Optional field (or constant) that gives the dimension of the alphanumeric array used as FROMFLD (e.g. in ALPHA/A60 (6) DIM would be 6).
TOPRUL/I	Optional ruler number to use when copying alpha-numeric fields to a text field. A ruler with the same number should exist in the Data Dictionary.
BOTRUL/I	Optional ruler number to end with when copying alphanumeric fields to a text field. A ruler with the same number should exist in the Data Dictionary.

The RMO should set W$W to 3 to force writeback of LINK records in TRANS after a call to TEXTCOPY, or else the copied text information in the TOFLD may not be available until EOFREC processing has taken place.

Be aware that if FROMFLD is alphanumeric, DIM = 0, TOPRUL = 0, and BOTRUL > 0, the TEXTCOPY call will have the effect of changing the top ruler in the text.

H.6.1.1 Using TEXTCOPY to set a Lock Mark

TEXTCOPY has a special syntax for setting the lock mark at the end of an internal text (TInn) field:

```
STAT = TEXTCOPY(MARK,TI_FIELD,OPTION)
```

Where

MARK/I	Mark to append. The only allowed value is 11
TI_FIELD/TI	Internal text field to append the Lock Mark in.
OPTION/I	Must be set to 1.

H.6.1.2 TEXTCOPY Examples

In the following code excerpt TEXTCOPY is used to insert the string "Explanation: ", which is stored in a local alphanumeric field at the top (OPTION = 2) of the text stored in the internal text field EXPLANATION (if its not already there!). The DIM argument is 1, as one line is to be inserted. TOPRUL is 3, specifying that Data Dictionary Ruler #3 is to be placed at the top of the file.

```
FILE: ERRMSG.MAS
LOCAL
.
EXPL_LAB/A13 'Explanation: '
EXPL_TEST/A13
TCSTAT/I
.
PROGRAM
.
LABEL_CHECK: ;
.
EXPL_TEST = STR(EXPL_TEST,EXPLANATION,1,13) ;
IF EXPL_TEST EQ EXPL_LAB THEN STOP ;
    ELSE TCSTAT = TEXTCOPY(EXPL_LAB,EXPLANATION,2,1,3) END
.
```

In this next code excerpt TEXTCOPY is used to copy the text stored in field SCR_EXPL to the field EXPLANATION. Any text already in EXPLANATION would be replaced.

```
FILE: ERRMSG.MAS
```

```
LOCAL
.
SCR_EXPL/TI60
TXSTAT/I
.
PROGRAM
.
TXSTAT = TEXTCOPY(SCR_EXPL,EXPLANATION,0)
```

The next code excerpt loads five lines at a time from the text field EXPLANATION into an array of alphanumeric fields.

```
FILE: ERRMSG.MAS
LOCAL
.
TCSTAT/I
ALPARR/A80(5)
J/I 1
.
PROGRAM
.
IF F$UNCKEY EQ 'F17' THEN GOSUB MORETEXT END
.
MORETEXT:  TCSTAT = TEXTCOPY(EXPLANATION,ALPARR,3,5,J) ;
           J = J + 5 ; RET
```

H.6.1.3 TEXTCOPY - Appending a Lock Mark at the End of Internal Text

The TEXTCOPY subroutine now appends a Lock Mark at the end of an internal text. The syntax is:

```
STAT = TEXTCOPY(MARK,TI_FIELD,OPTION)
```

where:

MARK/I	Mark to append. The only allowed value is 11.
TI_FIELD/TI	Internal text field in which to append the Lock Mark.
OPTION/I	Must be set to 1.

When copying an array of alpha fields to a TI field, TEXTCOPY can be instructed to act as if there is a hard C.R. at the end of each field (i.e. the text is not normalized). Add 4000 to OPTION to get this behavior.

TEXTCOPY can also be told to take its input from an ASCII file. Make the FROM field (1st argument) an An field containing the file name of the input text, and set OPTION = 8. If TEXTCOPY cannot open the file STAT is returned with -20.

To output internal text to an ASCII file make the TO field an An file containing the file name, set OPTION = 16, and if the internal text is in RTF format set the DIM field to the maximum line length you want in the output file. If the output file cannot be created STAT is returned with -21.

H.6.2 TEXTATTR: Get Text Field Attributes

Use the TEXTATTR subroutine to obtain the attributes (e.g. Number of lines, Date Last Modified, etc.) of the piece of text stored in the specified text field (field type TInn or TXnn).

Syntax:

STAT = TEXTATTR(TXTFLD|BLOBFLD,OPTIONS,FLD1,...)

STAT/I	
Value	Status
1	OK. Text attributes successfully loaded
0	TXTFLD contains no text
-1	TXTFLD is not a text field
-2	OPTIONS is not an alpha field
-3	More options than target fields
-4	Invalid editor code (only 2 or 3 may be used for RTF text)
-5	Invalid format (cannot change format unless text/ blob has code 2, 3 or > 256).
-6	Unable to write back new format (file may be opened Read Only)
-101	Target field 1 has incorrect data type
-102	Target field 2 has incorrect data type
-1xx	Target field xx has incorrect data type
-201	Option # 1 has unrecognized value
-202	Option # 2 has unrecognized value
-2xx	Option # xx has unrecognized value.
TXTFLD/TIn or TXn or BLOBFLD/BLOB	Text field or BLOB to get attributes for. Must be a field.
OPTIONS/An	(Field or constant) String made up of alpha codes that designate attribute(s) to be loaded into target field(s). If more than one attribute is requested, codes are concatenated together, separated by a "+" (plus) character. For each attribute requested, a corresponding target field of appropriate data type must be declared following the OPTIONS argument.
Valid Attribute Codes	(Target field type) Description
DATCRE	(DT) Date created
TIMCRE	(TM) Time created
DATMOD	(DT) Date last modified
TIMMOD	(TM) Time last modified
LOCKED	(L) Line # locked at
NLINES	(L) Lines in text
NCHARS	(L) Characters in text
FORMAT	(I) Text editor 1=ADMINS Internal(VMS) 2=AdmTed 3=MS Word >256=BLOB format
EDITOR	(I) Used to change the format (the default text editor or the BLOB format), e.g.: STAT = TEXTATTR(TFIELD,'EDITOR/A6',FMT) where the value in the FMT/I field will update the format attribute in the TCF record for the text field or BLOB. The only allowed values for FMT are 2 and 3 to set which editor will be used for this text field, or a value > 256 when setting the format for a BLOB.

FLD1/xx	Field(s) to receive attribute value(s). There must be as many target fields as there are option codes in OPTIONS, and each field must have the correct data type for the attribute value it is to receive.

H.6.2.1 TEXTATTR Example

In the following example TEXTATTR is used to get the number of lines of text in TXTFLD, and the line number where TXTFLD is locked.

```
FILE TEXTFILE.MAS
LOCAL
.
ATTROPT/A20 'LOCKED+NLINES'
TXTLINES/L
LCKLINE/L
.
PROGRAM
.
STAT = TEXTATTR(TXTFLD,ATTROPT,LCKLINE,TXTLINES)
```

H.6.3 SEARCH: Find Character String in Text Field

The SEARCH subroutine finds the specified character string in the specified text field (TInn or TXnn). SEARCH is called using the following syntax:

```
FOUND =
SEARCH(TXTFLD,STR[,CASE[,LINE[,NLNS[,WRD[,DIREC[,METACH]]]]]])
```

FOUND/I	1 = String was found. 0 = String was not found -1 = Invalid # of arguments -2 = Invalid search direction (use 0 or 1) -3 = Invalid case sensitivity code (use 0 or 1)
TXTFLD/TIn /TXn	Text field, internal or external, to be searched.
STRING/An	(Field or constant.) Character string to be searched for in TEXTFIELD.
CASE/I	(Optional.) 0 = case insensitive search. 1 = case sensitive search.
LINE/L	(Optional.) Line # to start the search. If 0 or 1, start at beginning of text, if -1, start at end of text. Returns line number where STRING was found.
#LINES/I	(Optional.) Number of lines to search relative to LINE.

WORD/I	(Optional.) Start search within LINE at this word number. Returns the number of the word within LINE where STRING was found.
DIREC/I	(Optional.) 0 = search forward in file from LINE. 1 = search backward in file from LINE.
METACH/A1	(Optional.)Default meta character for wildcard search is '*' (asterisk). Provide this argument if you want to override the default meta character. If METACH is blank, no character is treated as a meta character.

Be aware that SEARCH does not work like a text editor search. A search for "WORD" using SEARCH will try to find "WORD" embedded in white space. (the beginning and end of a line is considered a white space, along with tabs, blanks, etc.). If you want to do a text editor like search for 'WORD' you would have to embed it in meta characters ("wildcards), i.e. "*WORD*". The various options are:

- WORD - Find 'WORD' embedded in white space
- *WORD - Find any word ending with 'WORD'
- WORD* - Find any word starting with 'WORD'
- *WORD* - Find any occurrence of 'WORD', regardless of surrounding characters.

  E.g., if your search string contained:

  'port* * *'

  SEARCH would pick up constructs like 'Port of Spain', 'ports in Spain', 'Portugal and Spain', etc.

H.6.4 SUBSTITUTE: Replace character string with another string

The SUBSTITUTE subroutine can be used to replace a character string in an internal text field (or an Alpha field) by another character string. The syntax is:

```
STAT = SUBSTITUTE(FIELD,FROM,TO,OPTION)
```

Where:

FIELD/TI FIELD/An		An internal text (TI) or Alpha field to operate on
FROM/An		An An field containing the character string to be replaced.
TO/An		An An field containing the character string that should replace the character string in the FROM field if found.
OPTION/I		Options controlling the substitution:
	1	Change all occurrences (by default only the first occurrence is changed).
	2	The FROM text is case insensitive (by default only character strings that match exactly using the character case will be changed, e.g. "text" and "Text" does not match) .
	4	Change whole words only, i.e. do not change a match if it is part of a longer word.
	8	Process only TI format 2 (AdmTed RTF) and 3 (WORD RTF). On Windows, by default old TED WPT format text will be converted to RTF before a search for matching strings are performed. With this option set old style text is not changed.
	16	Convert old style TED WPT text to RTF even if no substitution took place. By default, if there is no match found in the field the field is left unchanged in TED WPT format.
STAT/I		Return status:
	>0	Number of occurrences changed
	0	No match found
	-1	Invalid number of arguments in subroutine call
	-2	FIELD is not TI (Internal text) or A (alpha)
	-3	An internal text field matched to a field in an external file (e.g. LINK file) does not match in data type
	-4	File is opened in Read Only mode, and cannot be changed.
	-5	Matching TI$FIELD not found (an internal error)
	-6	Unable to convert WPT to RTF format

H.6.5 EDIT: "Paragraph" Editing in TRANS

TRANS "Edit Mode" allows editing of individual characters within a field without retyping the entire field.[6] The EDIT subroutine expands this capability to allow editing of several consecutive alphanumeric fields in the main file of the screen as a group, or "paragraph". In this facility leading blanks are retained to support indentation of paragraphs.

When the EDIT subroutine is called the message

```
--------------------EDIT-------------------
```

appears on the screen directly above the group of fields to be edited as a paragraph, signaling that the EDIT subroutine is enabled and delimiting the width of the paragraph. This message disappears when the EDIT subroutine exits.

When the EDIT subroutine is enabled the following keystroke functions may be used in addition to the normal TRANS editing keystroke functions:[7]

KEYSTROKE	EDIT subroutine function (Paragraph Editing)
ed.ext	Leave EDIT subroutine
ed.dell	Delete the current line
ed.down	Move the cursor down one line, same column.
ed.insl	Insert a line after the current line*.
ed.norm	Normalize the paragraph from the current cursor position to the bottom, by putting as many words as possible on a line, once space between each.
ed.up	Move the cursor up one line, same column.

The last line of the "paragraph" must be empty in order to insert a line while the EDIT subroutine is enabled. If the last line is not empty TRANS sounds a warning tone and displays the message:

```
Last line NOT empty
```

Normalizing text will not remove leading blanks (indents are preserved) or blank lines.

H.6.5.1 EDIT Syntax

```
NULL = EDIT(FIELD,X,Y,NF,WIDTH)
```

NULL/I	Required for syntax purposes only.
FIELD/An	The first of a consecutive sequence of An fields in the main file of the screen, that are to be edited as a group, or “paragraph”.

6. See Section 6.4.1 “Keystrokes: Entering or Changing Fields”.
7. When the EDIT subroutine is enabled the "Show TRANS editing keys" display is extended to include the paragraph editing functions.

X/I	The line number of the upper left corner of the rectangle where the group of fields have been placed on the screen.
Y/I	The column number of the upper left corner of the rectangle where the group of fields have been placed on the screen.
NF/I	Number of fields in the group to be edited as a paragraph.
WIDTH/I	The width of the paragraph (the size of FIELD1 and the NF consecutive fields of the same size). WIDTH must be even, and less than or equal to 80.

H.6.5.2 EDIT Examples

Following are two examples of instruction files that use the EDIT subroutine. The first screen (TEXT) allows editing of a single paragraph, while the second (TEXT2) supports editing of two separate paragraphs.

```
*  TEXT.DEF
*
MAS 100
N    I    KEY1
L1   A30
L2   A30
L3   A30
L4   A30
L5   A30
L6   A30
L7   A30
```

```
*  TEXT.TRS
*
TEXT TEXT.MAS 1 TEXT.RMO NOMSG APPEND AUTOCR
E N
ER ED/A1  [4,19,1]
E L1
E L2
E L3
E L4
E L5
E L6
E L7
SCREEN
CE TEST OF TEXT EDITING IN TRANS
BL
N: N-
TYPE 'X' TO EDIT:
BL
L1----------------------------
L2----------------------------
L3----------------------------
L4----------------------------
L5----------------------------
L6----------------------------
L7----------------------------
END
```

```
*  TEXT.RMS
*
FILE TEXT.MAS
LOCAL
M$M/A2
S$S/A6
*
* FIELDS TO DESCRIBE POSITION OF
* PARAGRAPH TO BE EDITED
X/I 6
Y/I 1
NF/I 7
WIDTH/I 30
*
* LOCAL FIELD TO INDICATE INTENTION TO EDIT
ED/A1
*
* DUMMY FIELD REQUIRED FOR SYNTAX
NULL/I
PROGRAM
IF M$M EQ 'UP' THEN ;
   IF S$S EQ 'ED' AND ED NE ' ' THEN ;
   NULL = EDIT(L1,X,Y,NF,WIDTH) ;
   ED = ' ' END END
```

The second example allows editing on two different paragraphs.

```
*  TEXT2.TRS
*
TEXT2 TEXT.MAS 1 TEXT2.RMO NOMSG APPEND AUTOCR
E N
ER ED/A1 [5,55,1]
E L1
E L2
E L3
E L4
E L5
E L6
E L7
SCREEN
CE  TEST OF TEXT EDITING IN TRANS
BL
N: N-
BL
TYPE '1' TO EDIT FIRST PARAGRAPH, '2' TO EDIT SECOND:
BL
L1---------------------------
L2---------------------------
L3---------------------------
L4---------------------------
BL
BL
BL
L5---------------------------
L6---------------------------
L7---------------------------
END
```

```
*  TEXT2.RMS
*
FILE TEXT.MAS
LOCAL
M$M/A2
S$S/A6
*  FIELDS TO DESCRIBE POSITION OF FIRST PARAGRAPH
X1/I 7
Y1/I 1
NF1/I 4
*  FIELDS TO DESCRIBE POSITION OF SECOND PARAGRAPH
X2/I 14
Y2/I 1
NF2/I 3
*  NUMBER OF CHARACTERS IN A LINE FOR BOTH PARAGRAPHS
WIDTH/I 30
```

```
*  FIELDS USED FOR EDITING BOTH PARAGRAPHS
ED/A1
NULL/I
PROGRAM
IF M$M EQ 'UP' THEN ;
   IF S$S EQ 'ED' THEN ;
      IF ED EQ '1' THEN ;
         NULL = EDIT(L1,X1,Y1,NF1,WIDTH) END ;
      IF ED EQ '2' THEN ;
         NULL = EDIT(L5,X2,Y2,NF2,WIDTH) END ;
      ED = ' ' END END
```

H.6.6 GETMSG - Retrieving "Literal" Text From a File

The GETMSG subroutine retrieves "literal" text from a file. This feature makes it possible to store and maintain text strings in an ADMINS file rather than as literal constants in an RMO, thus avoiding having those character strings count against the internal constant array in the RMO.

To use this subroutine, you must define an ADMINS file where the first field is an I (integer) field, and the second field is an A80 field (the field names do not matter). The file may or may not contain additional fields, but only the two first fields in the file are used for this feature.

The first field contains a number that uniquely identifies the text in the second field.

Then, create a logical name ADM$MESSAGEFILE containing the full pathname of the file. Then GETMSG can be used to retrieve the text strings from the file. The syntax is:

```
STAT = GETMSG(ID,TEXT)
```

where:

ID/I	The unique number that identifies a certain text string. The number can be a field or a constant.
TEXT/An	An alpha field to receive the text string identified by ID.
STAT/I	1 = OK 0 = No such text id -1 = ADM$MESSAGEFILE not assigned or not found -2 = ADM$MESSAGEFILE has invalid format -3 = Unable to allocate memory for text strings

The first time GETMSG is called, it loads all the records in the ADM$MESSAGEFILE file into memory, and keeps the texts in memory for the rest of the TRANS session. This memory does not count against any of the limited memory arrays in TRANS.

The following subroutines will also retrieve messages from the ADM$MESSAGEFILE file if arguments are passed on the form: #nn# i.e. the character '#' followed by the message number followed by the character '#':

DLGBOX, MSGBOX and ASKSCR

H.6.7 PARAG: Reformat Consecutive Fields as Paragraph

PARAG reformats the data in a series of alphanumeric (An) ADMINS fields into a left justified paragraph. PARAG does not split words across field boundaries unless a word is longer than the field length.

```
STAT = PARAG(FIELD,NFLDS,OPTION)
```

FIELD/An:	Name of first An field in paragraph
NFLDS/I:	Number of fields in paragraph
OPTION/I:	0 if input words do not cross field boundaries 1 if input words cross fields
STAT/I:	-5: FIELD is not in file -4: Not all fields are in file -3: Invalid OPTION -2: fields in paragraph are not all same size or are not all An fields -1: not enough fields for normalized text (can occur when option is set to 1) 1: OK

PARAG handles input data in which words are assumed not to be split across field boundaries. It also has an option for data, such as that produced by ACQUIR, where input words may cross field boundaries and fields may have hat (^) characters to hold leading blanks. For example:

```
      NOT SPLIT (OPTION = 0)              SPLIT (OPTION = 1)
------------------------------     ------------------------------
This text was entered in           This text was ACQUIRed. Assume
a screen.                          ^that leading and trailing
Assume                             blanks in fields indicate gaps
that the                           ^between words, and
beginning of each line is the      leading hats indicate blanks.
beginning of a word.
```

The alpha fields on which PARAG operates must be fields in the DEF of the RMO file, the fields must all be the same size, and they must be in consecutive order. PARAG does the following:

1. Changes tabs to blanks.
2. If input words cross field boundaries (OPTION argument is 1), changes leading hats in input fields to blanks (in case data was created with ACQUIR 'B' option).
3. If words do not cross field boundaries (OPTION argument is 0), inserts a blank between the end of one field and the beginning of the next field. Otherwise, fields are joined end to end.
4. Squeezes out leading blanks, trailing blanks, and multiple blanks.
5. Writes the data back beginning with the first field, putting as many words as possible in each field without splitting words across field boundaries. The output may occupy fewer fields than the input did; unused fields at the end of the paragraph are blanked out.

If you are using option 1 (the input contains words split across field boundaries), the output data may occupy more fields than the input data. If using option 1, space for expansion (blank fields) should be provided at the end of the paragraph. The shorter your fields and the longer your words, the more extra fields you will need. The maximum number of extra fields needed for expansion equals the number of fields which contain input data. If there aren't enough fields for the output data, PARAG returns a status of (-1). If this occurs the RMO should not write back the record (use W$W), because if the record is written, data will be lost.

Another result of option 1 is that after joining the fields, a "word" may be longer than the field length. In this case PARAG splits the word across as many fields as necessary.

H.6.8 TED: Call TED Text Editor in TRANS

The TED subroutine lets you invoke the TED Text EDitor to view and edit a text editable file, or an array of ADMINS An fields. The syntax is:

```
STAT = TED(WH,TXT,COL,LN[,OPT])
```

WH/I	0 = Edit the text file named in TXT. 1 = TXT is the name of a field that is the base of an array of An elements.
TXT/An [(dim)]	If WH is 0, an An field that contains the name of a text file. If WH is 1, the base of an array of type An (n should always be an even number, i.e. A48, A56, A80)
COL/I	Maximum width of text. If WH = 1, COL must match the size of the Alpha field, i.e. if TXT is field type A60 COL must be 60. COL should always be an even number.
LN/I	Maximum number of lines to edit. If WH = 1 LN must be equal to or less than (dim), the size of the text array.
OPT/I	Optional 0 = Read only 1 = Edit (Read/Write)
STAT/I	>0 = OK (number of lines edited) -1 = Invalid function code (WH not 0 or 1) -2 = TXT not An type field -3 = COL and/or LN missing or invalid for WH = 1 -4 = COL < 20 or COL > 80 for WH = 1 -5 = Invalid syntax for file name -100 = Text not saved

TXT must be a field. WH, COL, and LN may be fields or numeric constants.

H.6.9 VIEWTEXT: Display Text File in TRANS

The VIEWTEXT subroutine provides the capability to display and/or edit a text file in a window on the TRANS screen. VIEWTEXT syntax is as follows:

```
STAT = VIEWTEXT(TFILE,LINE,COL,NL,NC)
```

where:

TFILE/An	Name of text file to display
LINE/I	Top screen line for text display box
COL/I	Left screen column for text display box
NL/I	Number of lines in window
NC/I	Number of columns in window
STAT/I	Return values are: 1 = OK 0 = File was empty -1 = File not found -2 = Syntax error or illegal window specification

When the RMO calls VIEWTEXT, the beginning of the specified file TFILE is displayed in a box whose size and location are defined by LINE, COL, NL and NC. (Specify NL and NC to be 2 greater than the actual display area you want, to allow for the horizontal and vertical borders.)

The following keystrokes are used to move around in the VIEWTEXT file:

1. The UP and DOWN arrows scroll up and down 1 line at a time.
2. ENTER displays the next line.
3. PREV and NEXT scroll up and down one "page" (3/4 of the scrolling region size.
4. NRECS prompts with "n:" and accepts the UP or DOWN arrows to go to the beginning or end of the file. Use HOME to cancel the request at the "n:" prompt. (All other responses to the "n:" prompt cause a warning tone.)
5. HOME exits from VIEWTEXT and return the user to the TRANS screen.
6. The RGHT and LEFT Arrows scroll right and left. If the text is wider than the specified window, the right side of the box enclosing the text is opened to indicate that there is more text to the right. Use the right arrow key to scroll the text sideways to the right. When the text is scrolled to the right, the left side of the box is opened to indicate there is more text to that side. Use the left arrow to scroll back to the left.
7. SEL prompts for a "Search string:". Enter the character string you want to find; the press LOOK to search for it starting at the current position in the file (the top line displayed), or press the UP arrow to begin the search at the beginning of the file. If the search text is found, the text will be highlighted.

   Once a search string has been entered you can find additional instances of the string by pressing LOOK again.
8. SHFK displays keystroke help.

Other keystrokes beep and do nothing.

The screen is normally refreshed after leaving the window: the box and the text disappear, and the portions of the screen that were overlaid are redisplayed.

The box containing the text can optionally be kept on the screen after VIEWTEXT exits. This is enabled by placing the "v" (lower case v) into the string assigned to the logical name OPTION (see Appendix A: "Options"). When OPTION 'v' is in effect the VIEWTEXT display will be printed out as part of the screen display with the PRT keystroke.

You may also invoke an editor to edit your VIEWTEXT text file from inside the VIEWTEXT window. Assign the name of an editor to the logical name ADM$TEXTEDIT, e.g.

```
$ ASSIGN "EDIT/EDT" ADM$TEXTEDIT
```

Then, if you press EDIT in the VIEWTEXT window, VIEWTEXT will then SPAWN the editor with the VIEWTEXT file as the file to be edited. Once you exit from the editor, you will be back in the VIEWTEXT window, displaying the newly edited version of the file.

VIEWTEXT supports File Open Dialog Box (see Appendix H.12.1.1 "File Open Dialog Box" for more information).

H.7 File Information Subroutines

The two subroutines FILE32 and FIELD allow for the capture of the file definition information stored in the header of an ADMINS file.

H.7.1 FILE32 - Retrieve File Information From File Header

FILE32 is used to access the file header of an ADMINS file and retrieve information as to the number of records in the file, the size (in blocks), number of fields, size of the key (in words), and last record and index positions used. When FILE32 is called with another file name, the current file is closed.

H.7.1.1 FILE32 Syntax

```
STAT = FILE32(FILNAM,NRECS,NBLOKS,NFLDS,KSZ,LSREC,LSIDX)
```

STAT/I	Set to 1 if the file is found and 0 if the file is not found.
FILNAM/An	Name of the file to be queried.
NRECS/D	Set to the number of records in the file.
NBLOKS/D	Set to the number of ADMINS blocks (1024 bytes) in the file.
NFLDS/I	Set to the number of fields in the file.
KSZ/I	Set to the size in words (1 word = 2 bytes) of the key in the file.

SLSREC/D	Set to the last record position used in the file.
LSIDX/I	Set to the last index block used in the file.

H.7.1.2 FILE32 Example

The file EXAMPLE.MAS has 10 records and the following file definition:

```
*  EXAMPLE.DEF
*
MAS 100
FLD1 I KEY1
FLD2 A16 - /LA
FLD3 D2 - /MAX
FLD4 F5
FLD5 DA
FLD6 XA9A9A9
```

Given the following fields and values,

```
STAT/I
FILNAM/A20 'EXAMPLE.MAS'
NRECS/D
NBLOKS/D
NFLDS/I
KSZ/I
LSREC/D
LSIDX/I
```

then

```
STAT = FILE32(FILNAM,NRECS,NBLOKS,NFLDS,KSZ,LSREC,LSIDX)
```

would result in the fields having the following values:

```
STAT/I     1
FILNAM/A20 'EXAMPLE.MAS'
NRECS/D    10
NBLOKS/D   13
NFLDS/I    6
KSZ/I      1
LSREC/D    61
LSIDX/I    1
```

H.7.2 FIELD - Retrieve Field Information From File Header

Once a file is opened via a call to FILE32, then the FIELD subroutine can be used to get information about its fields. FIELD would be called once for each field in the file if information about all fields is required. The information available is the field's name, type, and size (in words); whether it is a key field; its picture (if applicable), and its derivation operator (if it has one).

H.7.2.1 FIELD Syntax

```
NULL = FIELD(N,FNAM,TYPE,NWRDS,KEY,PICT,DER)
```

NULL/I — Required for syntax purposes only.

N/I — Number of the field that FIELD is to retrieve based on its ordinal position in the file.

FNAM/A24 — Set to the name of the Nth field.

TYPE/I — Set to code for field type of the Nth field, as follows:

Code	Field Type
1	Dn
2	Ln
3	I
4	DA
5	An
6	Picture
7	Fn
8	DT
9	TM

The type is placed in the right byte of TYPE. In the case of Dn or Fn, the number of decimal places is placed in the left byte of TYPE. For Ln, Dn or Fn fields, FIELD returns values for TYPE according to the following formula (where NDEC is the number of decimal places, and FTYP is "1" for Dn fields, "2" for Ln fields, and "7" for Fn fields):

```
TYPE = (256 * NDEC) + FTYP
```

That is, for file type D TYPE is 1, D1 is 257 ((256 * 1) + 1), F2 is 519 ((256 * 2) + 7), etc.

NWRDS/I — Set to the number of words occupied by the data for the Nth field.

KEY/I — Set to the key status of the Nth field. For example, 2 means the second key field. Negative values indicate descending keys, i.e. -2 means the second key field and a descending key. For non-key fields, KEY is zero.

PICT/A24 — Set to the picture for the Nth field, if applicable.

DER/I — Set to the derivation operator for the Nth field, if applicable. The operation codes are 1 to 9, corresponding to /V, /E, /AVG, /MAX, /MIN, /FI, /LA, /C and /SA. If no operator is present, DER is 0.

H.7.2.2 FIELD Example

The file EXAMPLE.MAS has been opened using the FILE32 subroutine (see Appendix H.7.1.2 "FILE32 Example") and has the following file definition:

```
*  EXAMPLE.DEF
*
MAS 100
FLD1 I KEY1
FLD2 A16 - /LA
FLD3 D2 - /MAX
FLD4 F5
FLD5 DA
FLD6 XA9A9A9
```

Given the following fields and values,

```
NFLDS/I 6
NULL/I
N/I
FNAM/A24
TYPE/I
NWRDS/I
KEY/I
PICT/A24
DER/I
```

then looping through the FIELD subroutine

```
NX: N = N + 1 ; IF N GT NFLDS THEN GOTO DONE END
NULL = FIELD(N,FNAM,TYPE,NWRDS,KEY,PICT,DER) ; GOTO NX
DONE: STOP
```

would result in the above fields having the following values for each field in the file EXAMPLE.MAS.

```
N  FNAM  TYPE  NWRDS  KEY   PICT   DER
-  ----  ----  -----  ---  ------  ---
1  FLD1     3    1     1            0
2  FLD2     5    8     0            7
3  FLD3   513    3     0            4
4  FLD4  1287    4     0            0
5  FLD5     4    1     0            0
6  FLD6     6    3     0   A9A9A9   0
```

H.8 Arithmetic Subroutines

This group of subroutines perform special purpose arithmetic functions.

H.8.1 CARITH - Perform Array Arithmetic on D Type Fields

The CARITH subroutine is provided to give the user the ability to perform a repeated arithmetic operation between two (2) data structures and store the result(s) in a third data structure for a given number of iterations in a **single** RMS statement. These data structures can be either arrays (local or in the DEF) or single elements, allowing for six (6) modes of use:

1. Many to many to many (Array to array with the result in an array)
2. Many to many to one (Array to array with the result a single element)
3. Many to one to many (Array to a single element with the result in an array)
4. Many to one to one (Array to a single element with the result in a single element)
5. One to one to one (Single element to single element with the result in a single element)
6. One to one to many (Single element to single element with the result in each element of an array)

For example, in a financial application, CARITH could be used to:

1. Subtract twelve (12) monthly expense figures from twelve (12) monthly revenue figures resulting in twelve (12) monthly profit figures.
2. Divide the single constant 1000 into twelve (12) monthly profit figures resulting in twelve (12) monthly profit figures in thousands.
3. Add twelve (12) monthly profit figures in a single field resulting in a single full year profit figure.

Although CARITH can be used to achieve reductions in RMO size and complexity by alleviating the need to write "loop" paragraphs, its major benefit is its speed. Testing has shown CARITH to be 10 to 15 times faster than the equivalent explicit statements in an RMO, depending on the arithmetic operation performed.

It is possible to use CARITH to change the elements of an input array. In this case the output argument is one of the input arguments. However, a single element should not be used as both an input and an output argument.

CARITH is meant to be used with D fields only. If fields that have decimal places (i.e. field type D1, D3 etc.) are used the decimal point is ignored, i.e. 3.67 is treated as 367. and 0.05720 is treated as 5,720.

H.8.1.1 CARITH Syntax

```
NULL = CARITH(OP,IN1,SB1,IN2,SB2,OUT,SB3,NELE)
```

NULL/I	Required for syntax purposes only.
OP/I	Arithmetic operation to perform. 1 means add IN1 to IN2. 2 means subtract IN2 from IN1. 3 means multiply IN1 times IN2 4 means divide IN1 by IN2 and round. 5 means divide IN1 by IN2 and truncate.
IN1/D	First input argument which can be either an array or single element.
SB1/I	Starting subscript for IN1. Note that for all subscripts (SB1, SB2, SB3), a subscript of 1 is used for the first element regardless of whether the array is a local array or from the data record. A subscript of 0 defines the argument to be a single element.
IN2/D	Second input argument which can be either an array or single element.
SB2/I	Starting subscript for IN2.
OUT/D	Output argument which can be either an array or single element.
SB3/I	Starting subscript for OUT.
NELE/I	Number of elements to be processed in each of the arrays upon which CARITH is being performed. If only single elements are being processed NELE must be ‘1’ to perform correctly.

H.8.1.2 CARITH Example

Using the following file definition with the sample record shown, the TEST record maintenance will shown the use of CARITH in three different modes of use.

```
*   TEST.DEF
*
MAS 10                  Record Content
*                       --------------
KY A1 KEY1                   'A'
VAL1 D                        1
VAL2 D                        2
VAL3 D                        3
VAL4 D                        4
VAL5 D                        5
TOTAL D                       0

*   TEST.RMS
*
FILE TEST.MAS
LOCAL
ZERO/I    0
I/I
NELE/I
TVAL/D(5) 6 10 9 7 12
OP/I
CONS/D    2
NULL/I
DUMMY/D
PROGRAM
*
* Subtract the value found in each of the 5 fields
* (VAL1 thru VAL5) in the sample record from the values in each
* of the 5 elements in the local array TVAL and place
* the resulting values into VAL1-VAL5. (Results in
* column 'AFTER STATEMENT 1' in the table below)
*
OP = 2 ; I = 1 ; NELE = 5 ;
 NULL = CARITH(OP,TVAL,I,VAL1,I,VAL1,I,NELE)
*
* Multiply the values found in VAL3 thru VAL5 by the value
* found in CONS (2) and place the results in VAL3-VAL5.
* (Results in column 'AFTER STATEMENT 2' in table below)
*
OP = 3 ; I = 3 ; NELE = 3 ;
 NULL = CARITH(OP,CONS,ZERO,VAL1,I,VAL1,I,NELE)
*
* Sum the values in VAL1 thru VAL5 and place the result in
* TOTAL. (Result in column 'AFTER STATEMENT 3' in the
* table below)(Note the use of a dummy field for an
* input argument)
*
OP = 1 ; I = 1 ; NELE = 5 ;
 NULL = CARITH(OP,VAL1,I,DUMMY,ZERO,TOTAL,ZERO,NELE)
*
```

The following table shows the results of the above uses of CARITH.

Results in VAL1 thru VAL5 and TOTAL

	START	AFTER STATEMENT 1	AFTER STATEMENT 2	AFTER STATEMENT 3
VAL1	1	5	5	5
VAL2	2	8	8	8
VAL3	3	6	12	12
VAL4	4	3	6	6
VAL5	5	7	14	14
TOTAL	0	0	0	45

H.8.2 FARITH - Perform Array Arithmetic on F Type Fields

The FARITH subroutine is identical to the CARITH subroutine except that the arrays or single elements are field type F instead of D.

FARITH is meant to be used with F fields only. If fields that have decimal places (i.e. field type F1, F3 etc.) are used the decimal point is ignored, i.e. 3.67 is treated as 367. and 0.05720 is treated as 5,720.

H.8.2.1 FARITH Syntax

```
NULL = FARITH(OP,IN1,SB1,IN2,SB2,OUT,SB3,NELE)
```

NULL/I, OP/I, SB1/I, SB2/I, SB3/I, and NELE/I function the same as in CARITH.

IN1/F, IN2/F, and OUT/F function the same as in CARITH except they are F type fields.

H.8.3 DPOWER - Raise a D Type Field to a Power

Use the DPOWER subroutine to raise a Dn type field to a power.

Syntax:

```
A = DPOWER(B,C)
```

A/Dn	Result of B raised to the power of C. Both B and C can have decimal places and be fractions and/or be negative. DPOWER assumes that the result field has the same type as the first argument; the result is calculated to the accuracy of the first argument. Since C can be a fraction, DPOWER can take roots.
B/Dn	Number to be raised to the Cth power.
C/Dn	Power to raise B to. For example, given the following fields and values: A/D3 B/D3 1.75 C/D 2 then A = DPOWER(B,C) would result in the value "3.063" in the field A.

H.8.4 FPOWER - Raise a F Type Field to a Power

Use the FPOWER subroutine to raise a Fn type field to a power. FPOWER's syntax is identical to DPOWER (above) except that all fields are type Fn.

Syntax:

```
A = FPOWER(B,C)
```

H.8.5 RANDOM - Random Number Generator

The RANDOM subroutine generates a random number between 1 and a specified value.

```
RND = RANDOM(RANGE)

 RANGE/I      RANGE is a positive value supplied by the user.
                RANGE can be a field name or a constant.

  RND/I         A random number between one and range.
                If RND is 0 an error has occurred
               (i.e. RANGE LT 1, too many/too few arguments).
N DUMMY.MAS 1 RANDOM.RMO NOMSG
DR N
ER NUM/I
ER MONEY/D2
ER ACTION/I
DR RND/I
*
BOX DEFAULT
SCREEN
BL
                        +====================+
DW           DICE GAME
                        +====================+
BL
               ENTER # [1-6]: NUM-   MONEY : MONEY---
BL
               ROLL[1/0]: ACTION--   RANDOM #: RND---
END     *
*  RANDOM.TRS
*  ==========

*
*  RANDOM.RMS
*  ==========
FILE DUMMY.MAS
S$S/A6
M$M/A2
NUM/I
MONEY/D2
RND/I
ACTION/I
M$MSG/A40
M$LOC/I 12
STAT/I
*
PROGRAM
M$MSG = ' ' ;
IF M$M NE 'UP' THEN STOP END
IF S$S EQ 'ACTION' AND ACTION EQ 1 THEN RND = RANDOM(6) ;
 IF NUM EQ RND THEN
 STAT = FORMAT('Congratulations, You won $*',MONEY,M$MSG)
     ELSE
STAT = FORMAT('Too Bad, You lost $*, try again!',MONEY,M$MSG)
   END
 END
```

H.8.6 SEQINC - Sequential Number Generator

Use SEQINC to generate unique sequence numbers for any number of series with fully-controlled concurrent access by any number of users. SEQINC is implemented using a regular ADMINS data file, from which SEQINC reads the current sequence number, increments or decrements it by the specified amount, and then writes it back, all under full ADMINS concurrency control. The general format for the sequence number file definition is:

```
tab 100
SEQID   type    KEY1
SEQSUB  type    KEY2
SEQNO   F               ! Sequence number
SEQINC  F               ! Amount to increment/decrement
                        ! Contains '1' if you want to
                        ! increment by one.
SEQMAX  F               ! If >0 specifies max value
                        ! when this value is returned
                        ! SEQNO reset to zero
```

This general format allows for any number of different sequence IDs, and any number of series for a given sequence ID, e.g. you could have independent sequential numbers for each month of the year. The field names may be whatever you want, the only requirements are that the file must have two key fields (of any field type), followed by two F fields, the first of which contains the last sequence number given, and the second contains the amount to increment (decrement if negative) the sequence number field with for each call to the SEQINC subroutine. Optionally a third field of type F may be included to impose a maximum value on the sequential number (after the maximum value is achieved the SEQNO is reset to zero). If the SEQMAX field is zero SEQINC imposes no maximum.

The syntax of the SEQINC subroutine is:

```
STAT = SEQINC(FILENAME,KFLD1,KFLD2,TARGETFIELD)
```

where

STAT/I — Return values

1	OK
-1	Invalid number of arguments
-2	FILNAME has invalid type (not alpha)
-3	File FILENAME not found
-4	KFLD1 wrong type or length
-5	KFLD2 wrong type or length
-6	SEQNO or SEQINC not field type F
-7	TARGETFIELD has invalid type (must be I, L, D, F, X99..99 or An)
-8	New value of SEQNO exceeds capacity of TARGETFIELD
-9	Specified record not found

FILENAME/An — Name of file where sequence number field is located. The file name must not have any options appended to it, as this could break the concurrency control of the file.

KFLD1	Field identifying the sequence field we want to increment/manipulate. Must match the type and length of key field 1 in FILENAME
KFLD2	Identifies the specific subseries of SEQID we want, e.g. year. If there is only one occurrence of a given SEQID, may be blank. Must match the type and length of key field 2 in FILENAME.
TARGETFIELD	Field in the virtual record that is to receive the incremented value of the SEQNO asked for. Must have type I, L, D, F, X9..9 or An.

The following example uses SEQINC in a TRANS RMO to automatically generate a sequential number for both purchase orders and invoices in either of two years 1995 and 1996.

Assume file SEQDIR:SEQFILE.TAB contains the following records:

SEQID/A8	YEAR/X9999	SEQNO/F	SEQINC/F
INVOICE	1995	95001056	1
INVOICE	1996	96000000	1
PO	1995	950326	1
PO	1996	960000	1

Each user enters fields on the screen indicating whether a new purchase or invoice (SEQID) is to be created, and for which year (YEAR). When the YEAR entry is made, SEQINC will open SEQDIR:SEQFILE.TAB, get a lock on the record identified by SEQID and YEAR, increment the value of the SEQNO field in that record, write the new value back to SEQDIR:SEQINC.TAB, release the lock, and return the incremented value in the NEWNO field, converted to X9999999.

```
*EXAMPLE.RMS:
*
FILE CENTRAL:NEW_ENTRY.MAS
LOCAL
*
M$M/A2
S$S/A12
STAT/I
FILNAM/A24 'SEQDIR:SEQFILE.TAB'
SEQID/A8
INVNO/X9999999
ERROR/I
*
PROGRAM
IF M$M NE 'UP' THEN STOP ; END
IF S$S EQ 'YEAR' THEN ;
   STAT = SEQINC(FILNAM,SEQID,YEAR,NEWNO) ;
   IF STAT NE 1 THEN ERROR = 900 ELSE ;
      IF SEQID EQ 'INVOICE' THEN ;
          INVNO = NEWNO ELSE PONO = NEWNO END ;
      END ;
   END
...
```

If a user entered "INVOICE" and "1996", SEQINC would return 96000001 in the field NEWNO. The next user who entered "INVOICE" and "1996" would get 96000002 in NEWNO.

H.9 Logical Name and Symbolic Name Subroutines

The subroutines handle creation and translation of logical names and local symbols.

H.9.1 CRLOG - Create or Delete a Logical Name

The CRLOG subroutine is used to create or delete logical names. Logical names created with CRLOG remain effective for the full login session.

H.9.1.1 CRLOG Syntax

```
STAT = CRLOG(LOGNAM,VALUE,[TABLE])
```

STAT/I	Set to the return status code received from the system directive (OpenVMS).
LOGNAM/An	Logical name to which the VALUE is to be ASSIGNed. Maximum length is 80.
VALUE/An	Value to be ASSIGNed to the logical name. If VALUE is blank the logical name referenced by LOGNAM is DEASSIGNed. Maximum length is 80.
TABLE/A1	Optional. Field name or constant that specifies which logical name table to use: "S" for system table, "G" for group table, "P" for process table, or "J" for job table.

H.9.1.2 CRLOG Example

Given the following fields and values,

```
STAT/I
LOGNAM/A20 'DATA'
VALUE/A10 '_DBA1:'
```

then

```
STAT = CRLOG(LOGNAM,VALUE)
```

would create the logical name "DATA" with the value "_DBA1:". The equivalent DCL command is as follows:

```
$ assign _dba1: data
```

If the field VALUE had been blank when the CRLOG subroutine was called, then the logical name "DATA" would be deassigned. The equivalent DCL command is as follows:

```
$ deassign data
```

H.9.1.3 Special CRLOG Syntax for Modifying OPTION

The CRLOG subroutine has a special syntax to make it easier to modify the contents of the logical name OPTION (see Appendix A: "Options"). This syntax is enabled only when the first argument (the logical name) given to CRLOG has the value "OPTION".

If the logical name is OPTION then if the first character of the second argument (the "value") is '+', the rest of the value is added to the OPTION string.

```
LOGNAM = 'OPTION' ; VALUE = '+WA'
STAT = CRLOG(LOGNAM,VALUE)
```

In the above example, if OPTION contains 'VILO' before CRLOG is called it will contain VILOWA after CRLOG is called.

Similarly, if the logical name is OPTION then if the first character of the second argument (the "value") is "-" then the subsequent characters in that argument will be removed from the string assigned to the logical name OPTION.

```
LOGNAM = 'OPTION' ; VALUE = '-VO'
STAT = CRLOG(LOGNAM,VALUE)
```

In this example, if OPTION contains 'VILOWA' before CRLOG is called it will contain 'ILWA' after CRLOG is called.

H.9.2 TRLOG - Translate a Logical Name

The TRLOG subroutine is used to translate logical names.

H.9.2.1 TRLOG Syntax

```
STAT = TRLOG(LOGNAM,VALUE,[TABLE])
```

STAT/I	0 = no translation 1 = OK (logical name translated) 2 = VALUE not long enough, truncated When TABLE="R"(retrieve Registry Value) -1 = A valid HKEY value was not found -2 = No such registry key was found -3 = The value was not of type REG_SZ, REG_EXPAND_SZ or REG_DWORD.
LOGNAM/An	Logical name from which the VALUE is to be translated. Maximum length is 80. If the TABLE argument is "R" this argument contains the name of the registry key.
VALUE/An	Set to the value currently ASSIGNed to the logical name referenced by LOGNAM. VALUE will be blank if nothing is assigned to LOGNAM. Maximum length is 80. If the TABLE argument is "R" this argument contains the name of the registry value within the key, because this value will be changed by the call it needs to be set everytime such a call is made.
TABLE/A1	Optional. Field name or constant that specifies which logical name table to use: "S" for system table, "G" for group table, "P" for process table, or "J" for job table. If set to "R" TRLOG can retrieve named values from the registry . If the value of the optional third argument, TABLE, starts with a '+' TRLOG performs a 'full' translation of the logical name (i.e. possible logical names contained in a translation are also translated). The value of the TABLE argument can be just the '+', or, for example, '+P' for the process table.

H.9.2.2 TRLOG Examples

Given the following fields and values, and that "_DBA1:" is assigned to the logical name "DATA",

```
STAT/I
LOGNAM/A20 'DATA'
VALUE/A10
```

then

```
STAT = TRLOG(LOGNAM,VALUE)
```

would translate the logical name "DATA" and place the string "_DBA1:" into the field VALUE. The value of STAT would be set to "1".

To retrieve a registry value:

```
KEY/A80 'HKEY_LOCAL_MACHINE\HARDWARE\DESCRIPTION\System\CentralProcessor\0'
VALUE/A20 'Identifier'
TABLE='R'
```

then

```
STAT = TRLOG(KEY,VALUE,TABLE)
```

might return something like 'x86 Family 15 Model 2 Stepping 7' in the VALUE field.

H.10 Record and Field Access Subroutines

These subroutines provide various ways of finding, accessing, and creating ADMINS data records and virtual records.

H.10.1 DDATTR: Get Data Dictionary Attributes & Codelists

The DDATTR subroutine provides access to Data Dictionary element attributes and Data Dictionary codelist values.

The DDATTR syntax is:

```
STAT = DDATTR([DIM,]ATTR,FIELD,ATTR_VAL[,ATTR_VAL2...]
```

STAT/I Status Code

1 OK
-1 Too few arguments in call
-2 ATTR not an Alpha field or constant
-3 No Codelist File Active
-4 Too few arguments to receive all values
-5 Invalid attribute code
-6 Field has no DDID, or ADM$DD not assigned
-7 ATTR_VAL (return) field has wrong type for item, not an A60 field, or field to receive CLT_UAC value is not an A16 field
-8 1st argument is not INTEGER or ALPHA
-9 Field list is not alphanumeric
-10 Field list contains field name not in ADD
-502 Field does not ref. Codelist Table
-503 No Codelist Value found for this field
-504 External codelist does not have requested field (Description or UAC)
-505 Can't open external codelist

DIM/I If present, i.e. if the first argument is not an alpha field, then FIELD is the name of an array whose elements contain the names of the fields whose attributes are to be retrieved, and ATTR_VAL, ATTR_VAL2etc. are the names of arrays whose elements are to receive the attribute values for the entries in the FIELD array. If DIM is not present, the requested attributes for the hard-coded field name FIELD are loaded into fields ATTR_VAL, ATTR_VAL2 etc. DIM controls how many fields are retrieved, i.e. if the arrays are sized 20 elements each and DIM has a value of 5, attributes for the first five elements only are retrieved.

ATTR/An Contains mnemonics for the attributes to get (must be alpha field or alpha constant).

Mnemonic	Meaning
CLT_DESCR	Get description value from Codelist Table for code value in FIELD (A60).
CLT_UAC	Get User Action Code from Codelist Table for code value in FIELD (A16).
EL_FORMAT	Format of dictionary element FIELD (A20).
EL_WIDTH	Output width of element (integer value).
EL_JUST	Justification (L or R) for element (A2).
EL_DESCR	Description for element (A60).
EL_LABEL	Line Label for element (A20).
EL_HEAD1	Column Header 1 for element (A20).
EL_HEAD2	Column Header 2 for element (A20).

	EL_MISC	Miscellaneous notes/action attribute for element (A80).
FIELD		If DIM is present as the first argument, FIELD is the name of an array whose elements contain a list of the fields whose attributes are to be retrieved. If DIM is absent, FIELD is the (hard-coded) field name for which you want to get attributes.
ATTR_VAL		If DIM is present as the first argument, ATTR_VALn is the name of an array whose elements receive the n-th attribute values for the corrsponding fields in the FIELD array. If DIM is absent, ATTR_VALn is a field name to receive the n-th attribute value for FIELD. The data types must correspond to the data types of the attributes being obtained(e.g. A60 for CLT_DESCR, I for EL_WIDTH).

Put the mnemonics for the codelist attribute values into the ATTR field (use a plus sign to specify multiple values), e.g.:

```
ATTR/A40 'CLT_DESCR'          (to obtain codelist description)
         or
ATTR/A40 'CLT_DESCR+CLT_UAC' (to obtain codelist description
                               and user action code)
```

The number of ATTR_VAL fields or arrays specified must correspond to the number of attributes named in ATTR.

In the following example four attributes are retrieved for each of the six fields in the FLDARR array:

```
*
LOCAL
FLDARR/A20(6) 'B_NAME' 'B_ORG' 'B_ADDR1'
          'B_ADDR2' 'B_PHONE' 'B_FAX'
1ATAR/A50(6)
2ATAR/A50(6)
3ATAR/A50(6)
4ATAR/A50(6)
ATTR/40 'EL_LABEL+EL_HEAD1+EL_HEAD2+EL_MISC'
STAT/I
DIM/I 6
*
PROGRAM
*
STAT = DDATTR(DIM,ATTR,FLDARR,1ATAR,2ATAR,3ATAR,4ATAR)
```

H.10.2 RECOPN and RECIDX - Access Records in any File

Use the RECOPN and RECIDX subroutines to access records in one or more files in many different ways: to find records by key, to read, write, insert, delete or transfer records, and once a position in the file has been established, to move to the next or previous record from the current record.

H.10.2.1 RECOPN - Open Files for RECIDX

The RECOPN subroutine is used to open a file for access by RECIDX. RECOPN establishes which fields are to be used as key fields, which fields are to be used to transfer values between the virtual record and the record buffer of the file being opened, and which fields contain the new key values for record transfers.

RECOPN syntax:

```
STAT = RECOPN(OP,FNO,FLAGS,FILNAM,KFLDS,RFLDS,VFLDS,TKEYS)
```

The arguments are:

OP/I	(field or constant) Operation to perform. 1 = Open the file named in FILNAM. If open is successful, a file number is returned in FNO. 2 = New fields being used for KFLDS, RFLDS, VFLDS, or TKEYS for already-open file.
FNO/I	If OP = 1, returned with file number to use in subsequent calls to RECIDX and RECOPN if open was successful. If OP = 2, file number of already opened file to modify. (Must be a field.)

FLAGS/I	(field or constant) Argument must be present. 4096=Open for Write (overrides e.g. ADM$READONLY)
FILNAM/An	`(field or constant)` `Name of file to be opened.`
KFLDS/ An(dim)	An alphanumeric array containing the names of the fields in the virtual record that contain the key values to be used in a record operation. Those fields must match the type and length of the key fields in the file being opened. Must be terminated by a blank element.
RFLDS/ An(dim)	An alphanumeric array containing the names of the fields in the file being opened which are to be read from or written to. Must be terminated by a blank element.
VFLDS/ An(dim)	An alphanumeric array containing the names of the fields in the virtual record which hold the values which are to be read from or written to the fields specified in the RFLDS array. Must be terminated by a blank element. Read operations load values from the RFLDS fields into the VFLDS fields. Write operations load values from the VFLD into the RFLDS fields before the record is written.
TKEYS/ An(dim)	An alphanumeric array containing the names of the fields in the virtual record which contain the new key values to be used in a transfer operation. Those fields must match the type and length of the corresponding key fields in the file, and the fields named in KFLDS. Must be terminated by a blank element.
STAT/I	1 = OK -1 = Invalid OP code -2 = Invalid file number for OP = 2 (This file is not open). -3 = Too many open files (Max 10) -4 = Unable to open the file -5 = FILNAM is not an An field -6 = Unable to access KFLDS array -7 = KFLDS is not alphanumeric -8 = Unable to access RFLDS array -9 = RFLDS is not alphanumeric -10 = Unable to access VFLDS array -11 = VFLDS is not alphanumeric -12 = unable to access TKEYS array -13 = TKEYS is not alphanumeric -100n = KFLDS field n not found -200n = KFLDS field n not correct type or length -300n = RFLDS field n not found -400n = VFLDS field n not found -500n = RFLDS/VFLDS field n not correct type or length -600n = TKEYS field n not found -700n = TKEYS field n not correct type or length

H.10.2.2 RECIDX Syntax

Once a file is opened by RECOPN, RECIDX is used to find, read, write, delete, insert, and transfer records in that file.

RECIDX syntax:

```
STAT = RECIDX(OP,FNO,FLAGS)
```

The arguments are:

OP/I	(field or constant) Operation to perform. 0 = Find a record by key values. 1 = Find the next record. -1 = Find the previous record. 2 = Update (write) the record. 3 = Insert a record. 4 = Delete the record. 5 = Transfer the record, i.e. change the keys of the record with the key values of the fields in KFLDS to the values of the fields in TKEYS. 9 = Close file number FNO. 10 = Close all files opened by RECOPN.
FNO/I	File number for file returned by RECOPN. (Must be a field.)
FLAGS/I	(field or constant) Argument must be present. 0 = No optional behavior specified. 1 = With operations 4 (delete) and 5 (transfer),delete/transfer) the current record position. Do not do a find using the KFLD value prior to the delete/transfer. A record postion must be established (if none is established RECIDX will return -5 in STAT. Use this flag for deletes/ transfers when file has non-unique keys. 2 = Do not load values from the "closest match" record actually found if the key value searched for is not found (with OP=0). 4 = When reading the data file, if the key value being sought is found, do not transfer values from that record to the virtual record fields. +8 = When reading the data file, if the record with the key value being sought is locked, return -8 and do not read the record. If the record is available, lock the record and read it(transfer its values into the virtual record). 4096=Write (overrides e.g. ADM$READONLY) when FLAG=4096 used in RECOPN

STAT/I	1 = OK 0 = Record not found, or End of file/Top of file. -1 = Invalid OP code -2 = Invalid file number. -3 = Cannot write before read -4 = Cannot write readonly file -5 = Record not found for delete or transfer. -6 = TKEYS field does not match KFLDS field -7= Insert/transfer not allowed when using alternate index -8=(When FLAGS include '+8') Record is found but already locked by another user, values from record are not read into VFLDS..

A find by key operation (OP = 0) must be performed before the previous and next operations (OP = -1 or 1) are allowed. A record must be read (OP = 0, -1 or 1) before a record can be written (OP = 2). For find, next, and previous operations (OP = -1, 0 and 1), values from the RFLDS fields in the record being read are transferred to the VFLDS fields. For write operations (OP = 2), values from the VFLDS fields are moved into the RFLDS fields in the record before it is written to the disk.

If no record exists with the keys specified in a read operation (OP = 0); or if a previous operation (OP = -1) is attempted from a position at the top of file; or if a next operation (OP = 1) is attempted from a position at the end of file; RECIDX returns STAT = 0, and the KFLDS fields are set to reflect the keys of the record actually found (unless FLAG=2)

Unless FLAGS is set to 1, record deletion operations (OP = 4) use the key values in the KFLDS fields to locate a record and delete it. If a record with that key is not found, nothing is done, and STAT = -5 is returned.

Unless FLAGS is set to 1, record transfer operations, (OP = 5) use the key values in the KFLDS fields to locate a record, then transfer that record to the key values found in the TKEYS fields. If no record with the KFLDS keys is found, nothing is done, and STAT = -5 is returned.

(If FLAGS is set to 1 the current record position is deleted or transferred.)

Update, insert, delete, and transfer operations (OP = 2, 3, 4, 5) do not change the VFLDS or the KFLDS fields.

RECIDX can signal back when the record it is trying to read is locked by another user. This is done by adding 8 to the value in the FLAGS argument, where OP = -1, 0 or 1 (Previous record, Find record by key or Next record)

Also for OP = 4 (delete a record) and OP = 5 (transfer a record) you can use FLAGS=8 when you are supplying a key value, to check whether the designated record is locked (RECIDX will return -8 if the record is locked). There is no need to check the lock status if you are deleting/transferring the record currently in memory (FLAG =1) as it is already locked.

FLAGS = 8 will be ignored if the file is not opened -M.

When FLAGS=8, if the record is found and not locked by another user the record is read (or deleted or transferred) and STAT = 1 is returned.

If the record is found but locked by another user STAT = -8 is returned. The field values are not loaded into VFLDS.

No record position in the file is established after insert, delete, and transfer operations (OP = 3, 4, 5). No record position in the file is established if the record being sought is locked, when lock status checking is in effect (e.g. when FLAGS has 8 added to its value). To establish a record position, a new RECIDX find call (OP = 0) must be made.

By default, all files are opened read only. You must use the explicit "-M", "-S", etc. file access options to open a file for writing (see Section 19.1 “Modes of File Access”).

For operations that are not record transfers (not OP = 5), a single, blank, alphanumeric field may be used as TKEYS argument.

H.10.2.3 RECOPN, RECIDX Examples

The two examples that follow demonstrate various capabilities of RECIDX.

Example 1: Alternate Index Simulation

One possible application for the RECOPN and RECIDX subroutines is to simulate alternative index access to a file. The instruction files that follow demonstrate an application where a menu bar is used to choose between accessing records by master ID (MASID, the primary key of the main file) or by NAME (the simulated alternative index).

DEF for the main screen file:

```
CITY    A20
STATE   A2
ZIP     X99999
...    MASTER.DEF
----------
MAS 100
*
MASID   D   KEY1
NAME    A24
ADDR    A32
```

DEF for the index file:

```
NAMEIDX.DEF
-----------
IDX 1000
*
NAME    A24  KEY1
MASID   D
```

The TRS:

```
RECIDX.TRS
    -----------
    RECIDX1 MASTER.MAS 1 RECIDX1.RMO NOMSG INSERT DELETE
    INDEX NAMEIDX.IDX
    NAME
    MASID
    END
    ER  MASID
    ER  WNAME/A24
    ER  ADDR
    ER  CITY
    ER STATE
    ER  ZIP
    DR NEWID/D
    DR STAT/I [10,7,5]
    BAR 1 OPTTIONS=VISUAL
      NAMEIDX EXECUTE NI
      Access by name
      MASID   EXECUTE MI
      Access by Master ID
    BOX 2 1 3 80
```

```
    SCREEN
    BL
    BL
                              MASTER FILE UPDATE/BROWSE
    BL
    BL
    Master ID: MASID---   Name: WNAME------------------
                          Addr: ADDR---------------------------
                          City: CITY---------------- State: ST-
ZIP: ZIP--
    BL
    STAT:        NEWID: NEWID---
    BRANCHES
    0 RECIDX1 NEWID
    &&Selfbranch
 END
```

The RMS:

```
RECIDX.RMS
    ----------
    FILE MASTER.MAS
    LOCAL
    M$M/A2
    S$S/A6
    B$B/A2
    F$UNCKEY/A4
    NEWID/D
    WNAME/A24
    IDX/I 0
    *
    FNO/I
    FLAGS/I
    OPNOPR/I 1
    RECOP/I
    FILENAME/A20
    KFLDS/A18(4)
    VFLDS/A18(4)
    RFLDS/A18(4)
    TKEYS/A1
    STAT/I
    C$C/A6

    PROGRAM
    IF (S$S EQ 'BEGREC') AND (M$M EQ 'UP' OR 'IN') THEN ;
       WNAME = NAME ;           ! Load NAME into Work Name Field
       IF IDX EQ 1 THEN C$C = 'WNAME' END
       END
   IF M$M EQ 'NI' THEN ;          ! Requesting Access by Name Index
       FILENAME = 'NAMEIDX.IDX-RM' ;
      KFLDS(1) = 'WNAME' ;           ! Use Work Name field as key
       KFLDS(2) = ' ' ;          ! Space terminate
       RFLDS(1) = 'MASID' ;             ! Load from MASID field
       RFLDS(2) = ' ' ;          ! Space terminate
       VFLDS(1) = 'NEWID' ;      ! Load into NEWID field
    *
       IDX = 1 ;                 ! Tell Alternate Index in use
       IF FNO EQ 0 THEN ;
 STAT = RECOPN(OPNOPR,FNO,FLAGS,FILENAME,KFLDS,RFLDS,VFLDS,TKEYS)
;
        END
       RECOP = 0 ; GOSUB LOCATE ;
       STOP ;
       END
```

```
    ************************************************************
    * Reset to main key on MI or Insert Mode
    *----------------------------------------------------------
    IF M$M EQ 'IN' THEN IDX = 0 ;
       IF S$S EQ 'WNAME' THEN ;
         NAME = WNAME ; END ; STOP ; END
    *
    IF M$M EQ 'MI' THEN ;
       IDX = 0 ;
       C$C = 'MASID' ; B$B = 0 ; STOP ; END

************************************************************
    * Processing NEXT and PREV keystroke for Alternate Index:
    *----------------------------------------------------------
    IF M$M EQ 'FX' AND IDX NE 0 THEN ;
       IF F$UNCKEY EQ 'next' THEN ;
          F$UNCKEY = 'RET' ;
          RECOP = 1 ;           ! Ask for Next Record
          GOSUB LOCATE ;
          GOTO EOP ;
          END ;
       IF F$UNCKEY EQ 'prev' THEN ;
          F$UNCKEY = 'RET' ;
          RECOP = -1 ;          ! Ask for Previous Record
          GOSUB LOCATE ;
          GOTO EOP ;
          END ;
       END
************************************************************
    * Typing in the Work Name field:
*-----------------------------------------------------------
    IF (M$M EQ 'UP') AND (S$S EQ 'WNAME') THEN ;
      IF IDX EQ 1 THEN ;            ! If alternate Index, go find it
          RECOP = 0 ;
          GOSUB LOCATE ;
          GOTO EOP ;
          END ;
       NAME = WNAME ;           ! Change of Name field
       END
    EOP: STOP

************************************************************
    * LOCATE subroutine:
    *----------------------------------------------------------
    LOCATE: ;
    STAT = RECIDX(RECOP,FNO,FLAGS)
    B$B = '0'
    RET
```

If access by name is requested (you type into the WNAME field after selecting NAMEIDX in the menu bar) RECIDX is used to retrieve the master ID (MASID) for that name from the name index file, NAMEIDX.IDX, and places it in the local field NEWID, then the screen self-branches to the main file record with key NEWID.[8]

Example 2: Bulk record transfer using MAINT with key value.

This demonstration uses MAINT's key value feature in combination with RECIDX to accomplish a bulk transfer of records from one (partial) key value to another.

Given a file with these keys:

```
ITEM   A20 KEY1
SIZE   A10 KEY2
FINISH A10 KEY3
```

8. A more elaborate version of this application might use the EDFLDS subroutine (see Appendix H.13.7 "EDFLDS - Modify List of Editable Fields in TRANS") to modify the cursor order and determine which fields are editable or display-only when using different access indices.

Use this RMS with MAINT to change all the records with the specified ITEM name to the new item name.

```
*
* CHANGENAME.RMO used with MAINT
* specific KEY VALUE syntax
* "MAINT CHANGENAME KEY"
*
FILE PARTS.MAS-RX
*
STAT/I
ASTAT/I
AFNO/I
PROMPT/A16  'Enter new name: '
NEW_NAME/A20
*
KFLDS/A10(4) 'ITEM' 'SIZE' 'FINISH' ' '
TFLDS/A10(4) 'NEW_NAME' 'SIZE' 'FINISH' ' '
FFLDS/A20(1) ' '
VFLDS/A20(1) ' '
*
LITEM/A20
FLAGS/I 0
*
FILENAME/A20 'PARTS.MAS-M'
*
W$W/I 0           ! Stops maint from over-writing RECIDX changes
PROGRAM
*
IF ITEM NE LITEM THEN ;       ! Reprompt if new key
*
  ASTAT = ASKSCR(2,6,PROMPT,NEW_NAME,8) END
*
*
IF AFNO EQ 0 THEN ;           ! Open once
 STAT = RECOPN(1,AFNO,FLAGS,FILENAME,KFLDS,FFLDS,VFLDS,TFLDS) END
STAT = RECIDX(5,AFNO,FLAGS)   ! transfer to new key value
*
LITEM = ITEM                  ! reset for testing on next call
*
STOP
```

Here's what the MAINT looks like when it is run:

```
$ maint changename key
Operating on GAMMA$DKA0:[BD.TEST.RECIDX]PARTS.MAS;1
OK to continue? Y
15:48:48.85
Key value: LFLANGE
15:48:54.65
Enter new name: CORNERPIECE
15:49:14.50 12 records processed
Key value:
15:49:16.34 12 records processed
```

H.10.3 FNDTAB - Set Up Data for LODTAB

FNDTAB and LODTAB are companion subroutines designed to load data from one or more fields in a file into arrays. The files referenced by FNDTAB/LODTAB do not have to be identified as LINK files in the TRS, although they may be.[9] The beginning (and alternatively the end) of the run of records from which data is loaded are specified by key value. Each time a section of the file is to be loaded into one or more array(s), FNDTAB and LODTAB must both be called. Each FNDTAB call is effective for only one LODTAB call.

H.10.3.1 FNDTAB Syntax

The syntax of the FNDTAB subroutine is:

```
STAT = FNDTAB(LODFIL,NKEYS,BRKEY,BEGKEY1,BEGKEY2,..,
[ENDKEY1,ENDKEY2,..][,SELECT][,CLOSE][,POSITION])
```

STAT/I	Returns the status of the FNDTAB subroutine. 1 means call was successful 0 successful call, no exact key match (if NKEYS negative) -1 means load file was not found -2 means NKEYS is not between 1 and 9 -3 means BRKEY is not between 1 and NKEYS -5 means no record had BEGKEYs (start key values) listed -6: Error parsing SELECT statement -7: Error adding ~FIELD in SELECT statement -8: Error compiling SELECT statement -(10+N) means the Nth key field doesn't match in field type
LODFIL/An	The name of the file from which data is to be read.
NKEYS/I	The number of key fields being used to find the first record to be loaded. If a file has four key fields, the value of NKEYS (positive or negative) can be 4 or fewer. If the value of NKEYS (positive or negative) is less than the actual number of key fields in the table file, the selection will be made on a partial key. NKEYS cannot be larger than (plus or minus) 9. IF NKEYS is negative, FNDTAB will load the next record after the indicated key values, if there is not an exact match. Otherwise, FNDTAB returns an error status if no record is found with the indicated key values.

9. The SHORT keyword (see Section 5.3.1.19 "SHORT: Conserve MD Array Space") may not be used if FNDTAB/LODTAB refer to a file that is also a LINK file in the screen.

BRKEY/I — Indicates the key field which contains the break key. Successive records will be loaded until the value of the key fields up to the BRKEY position changes. If BRKEY is 1, then the first key field is the break key, if BRKEY is 2, then the second key field is the break key, etc. If BRKEY is 0 (zero), then there is no break on key change and the number of records loaded will equal NVALS in the subsequent LODTAB call (see below).

If BRKEY is negative, then FNDTAB will load a range of keys, using the magnitude of BRKEY to determine the number of ENDKEYs (finish key values) to use. If "-BRKEY"(i.e. the magnitude of BRKEY if negative) is less than the number of keys in the table file then the end of the key range is determined by a partial key.

BEGKEYs — The key values specifying the beginning of the range of records to be loaded. The number of BEGKEYs must equal NKEYS. The field types of the fields specified by BEGKEYs must match the field types of the key fields in the table file specified by LODFIL.

ENDKEYs — Optional. Used when BRKEYs has a negative value to specify the key values of the end of the range of records to be loaded. The number of ENDKEYs must equal the magnitude of BRKEYS. The field types of the fields specified by ENDKEYs must match the field types of the key fields in the table file specified by LODFIL.

SELECT/An(d) — Optional. Alphanumeric field or array containing a SELECT expression. If the select statement spans fields a ':'(colon) must be used as a continuation indicator (similar to SELECT statements spanning lines in DEFINE or REPORT).

If the expression references a field in the virtual record a '~' (tilde) must prefix the field name (e.g.

```
TRANSDATE GT ~STARTDATE
```

where TRANSDATE would be in the LODFIL file, and STARTDATE would be in the RMOs virtual record.

If SELECT is not used it can simply be omitted (if the argument in this position is alphanumric it is assumed to be a SELECT expression, otherwise it is assumed to be the CLOSE argument).

CLOSE/I — Optional. Enables the RMO to control when a FNDTAB file is closed. If present, CLOSE must be the last argument in the FNDTAB call, or immediately precede the POSITION argument. If CLOSE is set to 1, then the file opened by FNDTAB is closed by the next LODTAB call. If CLOSE is set to 2, the next LODTAB call does not close the file: instead, the next time FNDTAB is called, if it needs to open a new file, it closes the previous file before opening the new one. If CLOSE has any other value, or is absent, then the file is not closed until the next branch.

POSITION/I	Optional. Can be set to a negative or positive number of records. If POSITION is set, the next LODTAB call uses its magnitude and moves up (negative, toward top of file) or down (positive, toward end of file) that number of records before returning values. POSITION can be used, for example, to return a set of values surrounding some key value: to obtain values from the record where KEY = 100 and the 5 records on either side of it, the FNDTAB call would use POSITION = (-5) and the LODTAB call would use NVALS = 11. POSITION must be an integer (I) field, not a constant. If used, POSITION must be the last argument, and the CLOSE argument must be present (if CLOSE isn't needed, set it to zero).

H.10.4 LODTAB - Load Data Into An Array Based On FNDTAB

LODTAB should be called immediately after a successful return from FNDTAB. FNDTAB and LODTAB must be invoked in the same RMO call. LODTAB can be called only once for each successful FNDTAB call.

H.10.4.1 LODTAB Syntax

The syntax of the LODTAB subroutine is:

```
STAT = LODTAB(FLDNAMES,NVALS,[AR1,AR2,...]|[,ADM$LODTAB])
```

STAT/I	Returns the status of the LODTAB subroutine. N means successful LODTAB call, N records loaded -1 means previous FNDTAB call was unsuccessful -2 means no previous FNDTAB call -3 means number of fields does not match umber of arguments -7 means two consecutive LODTAB calls -(10+N) means Nth field not in LODFIL -(30+N) means field type mismatch in Nth field between argument field and LODFIL -(100+N) means Nth field in ADM$LODTAB not found -(200+N) means field type mismatch for Nth field in ADM$LODTAB
FLDNAMES/ An(n)	An array of the field names to be loaded. The last element in FLDNAMES must contain a blank character, i.e. ' '. Each field named must be present in the table file.
NVALS/I	Up to NVALS values will be loaded into the arrays. Loading stops when the key range has been satisfied or when the number of records loaded equals NVALS, whichever comes first.

AR1,...	The array(s) into which the data contained in the specified fields from the selected records is loaded. The arrays must match the field types of the fields specified in FLDNAMES. (They may have different names.) The field in the first element of FLDNAMES will be loaded into the first array, the field in the second element of FLDNAMES will be loaded into the second array, etc. Each array must have at least NVALS elements.
ADM$LODTAB/ An(n)	Alternatively, the arrays that are to receive values may be specified via an array with the reserved field name (or prefix)[a] ADM$LODTAB. This syntax allows more fields to be loaded in a single FNDTAB/LODTAB call. The number of local array field names in ADM$LODTAB must exactly match the number of field names in FLDNAMES. The data types and lengths of the corresponding fields in the two arrays must match. ADM$LODTAB array(s) must be terminated by a blank element.

a. Any field name that begins with the string "ADM$LODTAB" (e.g. ADM$LODTABXYZ, ADM$LODTAB4) can be used to specify the list of local arrays that are to be loaded by LODTAB (which allows you to specify any number of FNDTAB/LODTAB calls by this method.)

H.10.4.2 FNDTAB And LODTAB Example

If the definition (DEF) of the table file contains:

```
* PAYRATE.DEF
TAB 500
DEPT     A10  KEY1
RANK     XA99 KEY2
YRS_RANK I
SALARY   D2
```

and the transaction screen (TRO) contains:

```
LINK PAYRATE.TAB-R
K DEPT
KC RANK
L YRS_RANK
END
(etc.)
```

The record maintenance procedure (RMO) contains:

```
LOCAL
S$S/A6
M$M/A2
STAT/I
LODFIL/A20 'PAYRATE.TAB'
NKEYS/I   1
BRKEY/I   1
DEPTX/A10 'CHEM'
FLDNAMES/A20(4) 'RANK' 'YRS_RANK' 'SALARY' ' '
NVALS/I   5
ADM$LODTAB/A18(4) 'RANKS' 'YEARSRNK' 'SALARY' ' '
RANKS/X99(5)
YEARSRNK/I(5)
SALARY/D(5)
.
.
```

```
.
(etc.)
PROGRAM
IF M$M NE 'UP' THEN STOP END
STAT = FNDTAB(LODFIL,NKEYS,BRKEY,DEPTX)
IF STAT NE 1 THEN GOTO ERROR END
*
STAT = LODTAB(FLDNAMES,NVALS,ADM$LODTAB)
IF STAT LT 0 THEN GOTO ERROR END
(etc.)
```

Note that the field named SALARY does not have to be specifically linked in, e.g. 'L SALARY', to be loaded into an array by LODTAB. The array which holds the data from the field RANK must have a field name other than RANK; RANK is a key to the link file and is used to make the link, and therefore must be renamed. The array holding the data from the field YRS_RANK is also renamed to avoid confusion between the array and the linked in field in the screen. SALARY is not linked into the screen and therefore the array can have the same name.

In this example the ADM$LODTAB reserved field name is used to specify the local arrays to be loaded by LODTAB. The LODTAB call could also have been specified as follows:

```
STAT = LODTAB(FLDNAMES,NVALS,RANKS,YEARSRNK,SALARY)
```

If PAYRATE.TAB contained data such as:

```
DEPT     RANK      YRS_RANK     SALARY
----     ----      --------     ------
.
.
.
ADMIN     35          5         39,775
CHEM      10          4         68,955
CHEM      15          8         61,340
CHEM      20          6         55,240
CHEM      20          4         49,995
ENG       10          9         78,335
.
.
.
(etc.)
```

After the RMO call above had been executed, there would then be the following values in the three arrays listed below:

```
RANKS/X99(5)      10        15        20        20

YEARSRANK/I(5)     4         8         6         4

SALARY/D(5)     68,955    61,340    55,240    49,995
```

The result of the FNDTAB and LODTAB calls is to load data into three arrays: RANKS, YEARSRNK, and SALARY. Each array has 4 values. The array in the field RANKS contains the data from the field RANK from the first five (or fewer) records with a primary key of 'CHEM' in the field DEPT. The array YEARSRNK contains data from the YRS_RANK field in the same records, and the array SALARY contains data from the SALARY field in those records.

Note that only 4 records are actually loaded. This is because only 4 records were found before the value in the field specified by BRKEY changed. The STAT return from LODTAB indicates the actual number of records loaded, in this example STAT is 4.

Note that if the following

H.10.5 GETFLD: Read Field Identified in Data Field

The GETFLD subroutine reads a field name from a data field, retrieves the value from the designated field, and writes that value to another specified field. If the two fields are not the same type, GETFLD will convert[10] the designated field's contents into the field type of the output. This subroutine enables the developer to specify an operation transferring information to a pre-determined field from a field **to be determined at run time.**

```
STAT = GETFLD([VALUE,]FLDNAM)
```

STAT/I	A status code returned when the subroutine is called. 0: If called with only one argument, the field named in FLDNAM contains a null value >0: The operation was performed. 1: If called with only one argument, the field named in FLDNAM contains a non-null value -1: The field named in FLDNAM was not found. -2: The contents of the field named in FLDNAM are not compatible[a] with the field type of VALUE.
VALUE/	A field of any type. VALUE will receive the contents of the field designated in the FLDNAM argument.

FLDNAM/An	Contains a valid field name in the file being operated upon. The contents of the field named in FLDNAM are written to VALUE. (The named field may be an actual field in the file, a virtual field in the screen, or a local field in the RMO). FLDNAM cannot be a constant.
	
	If this is the only argument present, GETFLD checks if the field name passed in the argument contains a NULL value or not.
	Syntax:
	
	`STAT/I`
	`FIELD/A18`
	`...`
	`FIELD = 'DISCOUNT'`
	`STAT = GETFLD(FIELD)`
	
	If the field DISCOUNT contains a null value (either zero or blank, depending on data type) STAT is returned as 0 (zero). STAT is set to 1 (one) if DISCOUNT contains a non-NULL value.

a. For example, an integer field cannot receive a value containing non-numeric characters.

10. Conversion is done internally via a call to NCAT (see Appendix H.3.1 "NCAT - Concatenating fields").

H.10.6 PUTFLD: Write Field Identified in Data Field

The PUTFLD subroutine provides the complementary function to GETFLD. PUTFLD reads a field name from a data field, and writes a value from a predetermined field into that designated field.

PUTFLD has a similar syntax to GETFLD, except that PUTFLD reads from the predetermined field (VALUE) and writes to a field specified at run time (the value in FLDNAM is the name of the field to be written).

H.10.6.1 GETFLD and PUTFLD Example

Given the following file definition, screen instruction file, and record maintenance procedure:

```
BUDGET.DEF
----------
 MAS 100
 DEPT X999 KEY1
 JAN D
 FEB D
 MAR D
 APR D
 MAY D
 JUN D
 JUL D
 AUG D
 SEP D
 OCT D
 NOV D
 DEC D
```

```
BUDGET.RMS
-----------
FILE BUDGET.MAS
LOCAL
STAT/I
MONTH/A10
AMOUNT/D
M$M/A2
S$S/A6
PROGRAM
IF M$M NE 'UP' THEN GOTO OUT END
IF S$S EQ 'BEGREC'
   THEN MONTH = ' ' ; AMOUNT = 0 ;
   GOTO OUT END
IF S$S EQ 'MONTH'
   THEN STAT = GETFLD(AMOUNT,MONTH) ;
   GOTO OUT END
IF S$S EQ 'AMOUNT'
  THEN STAT = PUTFLD(AMOUNT,MONTH) END
OUT: STOP
```

```
BUDGET.TRS
----------
BUDGET BUDGET.MAS 1 BUDGET.RMO
E DEPT
ER MONTH/A10
ER AMOUNT/D
SCREEN
Department Monthly Budget Updates
BL
Enter Department: DE-
                  ---
Budget for Month of MO- : $ -----AMO
                     ---     --------
END
```

When a field name (i.e. JAN, FEB etc.) is typed into the virtual field MONTH, GETFLD is used in the RMO running behind the screen to place the value for that field in the virtual field AMOUNT. If a value is entered in AMOUNT (to update the budget record) PUTFLD is used to write to the new value to the field specified in the MONTH field.

For example, if a user enters "106" in DEPT, then enters "FEB" in the MONTH field, the screen will display in the AMOUNT field the budget value stored in the FEB field for the record with a key of "106". Should the user update the value in AMOUNT the change will be made in the FEB field of the "106" record.

H.10.7 OUTPUT - Append Records to Data File

The OUTPUT subroutine appends records to a data file, using data in RMO fields (usually arrays) which have the same names and types as fields in the output file. OUTPUT is similar to the OUTFILE feature in MAINT (see Section 10.12 "Writing Other Files: OUTFILE/OUTRECS").

The OUTPUT syntax is:

```
STAT = OUTPUT(FILENAME,NRECS)
```

STAT/I	Status Code 1 OK, records appended -1 FILENAME blank, nothing done -3 Records appended, but there was a field type mismatch between one or more pairs of fields with the same names. This may or may not be an error. -4 Cannot open output file using alternate index, nothing done
FILENAME/ An	Output File Name, must be a field
NRECS/I	Number of records to append, can be a field or a constant. If NRECS is less than zero, OUTPUT appends (-NRECS) records to the file, and leaves the file open*. If NRECS is equal to zero, OUTPUT just closes the file.

OUTPUT exits with a message and fatal error status if (1) the output file in FILENAME cannot be opened; or (2) there are no fields in the output file which have the same names and types as fields in the RMO.

When OUTPUT is asked to append more than one record, the fields to be transferred must be in arrays; and the arrays must have at least as many elements as the number of records to be appended. In the example below, data in the AVG array is transferred to the AVG field in the output file. 5 records are appended, and the AVG array is dimensioned with 5 elements. When OUTPUT is appending only one record, the data to be transferred can be in any field (need not be an array).

By default, OUTPUT opens and closes the output file each time it is called. Therefore, different OUTPUT calls can append records to different files. If the NRECS argument is negative, OUTPUT leaves the file open[11] after records are appended, which saves the work of closing the file and re-opening it if more records are to be appended with subsequent calls to OUTPUT.

OUTPUT does not append records when the RMO is run in test mode.

If the output file has alternate indexes, they wil be maintained. That is, the record is inserted instead of being appended.

By default, OUTPUT opens the output file in EXCLUSIVE mode.

An example of an RMS which calls OUTPUT:

11. OUTPUT can only have one file open at a time. If OUTPUT is asked to open a new file while a file remains open from a previous OUTPUT call, the previously opened file is closed before the new file is opened.

When OUTPUT is called, it matches fields in the RMO with fields in the output file AVG.MAS. The AVG field exists in both, and has the same field type. OUTPUT is asked to append 5 records to AVG.MAS, so it puts the value of AVG(1) in the AVG field of the first output record, AVG(2) in the second record, etc. The field 'N' in AVG.MAS does not exist as an RMO local field or as a field in MAIN.MAS, so 'N' is set to zero in the output records.

```
* AVG.RMS
 * -------
 FILE MAIN.MAS
 LOCAL
 STAT/I
 FILENAME/A30 'AVG.MAS'
 AVG/D2(5)
 NUM/D2(5) 100 200 300 400 500
 I/I
 PROGRAM
 I = 0
 LOOP:  I = I + 1 ;
        AVG(I) = NUM(I) / 2 ;
        IF I LT 5 THEN GOTO LOOP END
 STAT = OUTPUT(FILENAME,5)

 * AVG.DEF
 * -------
 MAS 100
 *
 N   I   KEY1
 AVG D2
```

In this example, AVG is the only field which exists in both the RMO and the output file, so only the AVG data is transferred. However, OUTPUT can transfer any number of fields: for example, if NUM/D2 was in AVG.MAS, both AVG and NUM would be transferred into AVG.MAS.

H.10.8 EDITMASK

The EDITMASK subroutine allows you to change Edit Masks within the Data Dictionary should the need arise. The syntax for this subroutine is as follows:

```
STAT = EDITMASK(FIELD,OPER,MASK)
```

where:

STAT/I	1 OK -1: FIELD has wrong data type -2: OPER has invalid value -3: MASK has invalid data type -4 - -8: Invalid edit mask
FIELD	If OPER is 1 thru 5 this argument is the field for which we want to change/add an edit mask. The data type of this field must be **A** or **X**. If OPER is 11 thru 15 this field contains the name of the field that is to have its edit mask changed (and thus it must be of field type **A,** and the field it identifies must be field type **A** or **X)**

OPER/I	1 or 11= Change (or add) an edit mask 2 or 12= Restore the default edit mask 3 or 13= Remove edit mask 4 or 14= Edit edit mask 5 or 15= Edit edit mask, and apply it
MASK/AN	An **An** field containing a valid edit mask if OPER = 1. Its content does not matter if OPER = 2 or 3. If OPER = 4 or 5, it is used as the starting edit mask if not blank, and contains the edited edit mask on return.

Example:

If the field ACCOUNT has an edit mask, and the field MASK contains the edit mask, the two calls below will have the same result:

```
STAT = EDITMASK(ACCOUNT,1,MASK)
```

and

```
FNAME = 'ACCOUNT'    ! FNAME/A18
STAT = EDITMASK(FNAME,11,MASK)
```

H.10.9 SUBFIELD: Obtain information about a field's sub-fields

The SUBFIELD subroutine obtains information about the sub-fields of a field. The general syntax is:

```
STAT = SUBFIELD(OP,ARG2,ARG3 [,ARG4 ...])
```

where:

OP/I	Operation to perform. The following operation codes are defined: 0: Get list of sub-field names 1: Copy an account and replace sub-fields 2: Check if a subfield is NULL (blank or zero) 3: Build parent value from subfields 4: Populate subfield values from parent field
ARG2 etc.	Depends on which operation to perform.
STAT/I	Return status. >= 0: See specific operation 0: No sub-fields found for this field -1: Invalid operation (OP not = 0) -2: Invalid number of arguments

H.10.9.1 SUBFIELD - Operation 0: Get a list of Sub-field names

Syntax:

```
STAT = SUBFIELD(0,PARENT,SUBFLDS)
```

where:

PARENT	The field for which you want to obtain sub-field information.
SUBFIELDS/ An (dim)	An array of alphanumeric (An) fields to receive the field names of the sub-fields.
STAT/I	Return status. > 0: Number of sub-field names returned 0: No sub-fields found for this field -3: The SUBFLDS argument is not a An array -4: The SUBFLDS array does not contain enough elements to receive all the sub-field names. -5: Unable to get name of sub-field. Most likely a dictionary error, e.g. pointing to the wrong dictionary.

H.10.9.2 SUBFIELD - Operation 1: Copy a Parent field and replace Sub-fields

Syntax:

```
STAT = SUBFIELD(1,SRCPARENT,TGTPARENT,SUBNAMES,SUBVALUES)
```

where:

SRCPARENT	The field source for the copy operation.
TGTPARENT	The target field for the copy operation. SRCPARENT is copied to TGTPARENT.
SUBNAMES/ An(dim)	An array of alphanumeric (An) fields containing the name(s) of the sub-fields in TGTPARENT being replaced.
SUBVALES/ An(dim)	An array of alphanumeric (An) fields containing the name(s) of the array(s), which contain the values of the sub-fields being replaced in TGTPARENT.
STAT/I	Return status. 1: OK 0: TGTPARENT has no subfields. -3: SUBNAMES array is not of type Alpha -4: SUBVALUES array is not of type Alpha -5: SUBNAMES array has more elements than SUBVALUES array. -6: The result has invalid data type for TGTPARENT -10n: Subfield n not found in TGTPARENT -20n: Value field n not found -30n: Error converting data for value field n

H.10.9.3 SUBFIELD - Operation 2: Check if a Sub-field is NULL (blank or zero)

There are three possible syntax formats for this check.

Syntax 1:

```
STAT = SUBFIELD(2,SUBFIELD)
```

where:

SUBFIELD/ An	An alpha field containing the name of the subfield to check in the form 'PARENT.SUBFIELD'.

Syntax 2:

```
STAT = SUBFIELD(2,PARENT,CHILD)
```

where:

PARENT/An	An alpha field containing the name of the parent field.
CHILD/An	An alpha field containing the name of the sub-field to check.

Syntax 3:

```
STAT = SUBFIELD(2,PARENT,SUBF#)
```

where:

PARENT/An	An alpha field containing the name of the parent field.
SUBF#	An integer field containing the sub-field number to check.

In all cases, STAT is returned with one of the following codes:

STAT/I	1: The sub-field contains a non-null value. 0: The subfield contains all null values (zeros or blanks). -2: Wrong number of arguments -3: The second argument is not of type An. -4: The subfield was not found. -5: Third argument has invalid data type. -6: The PARENT field has less subfields than the number specified in the SUBF# field.

H.10.9.4 SUBFIELD - Operation 3: Create a parent field value from subfield values

Syntax 1:

```
STAT = SUBFIELD(3,PARENT)
```

Syntax 2:

```
STAT = SUBFIELD(3,PARENT,CHILDREN)
```

where:

PARENT/*	The name of the parent field for which to assemble a value.
CHILDREN/ An (dim)	An alpha array containing the names of the subfields to use. These may be real subfields of the parent field (e.g. ACCNT.FUND) or fields with the same data type as the corresponding subfields. If the CHILDREN array is not present, the actual subfields of the parent field are used.

STAT is returned with one of the following codes:

STAT/I	1: OK -2: Wrong number of arguments -3: The subfield argument is not an An array -5: No subfields found for the parent field -6: The concatenated result is not a valid value for the parent field -(10+n): Subfield n not found in parent field

H.10.9.5 SUBFIELD - Operation 4: Populate subfield values from parent value

Syntax 1:

```
STAT = SUBFIELD(4,PARENT)
```

Syntax 2:

```
STAT = SUBFIELD(4,PARENT,SOURCE)
```

where:

PARENT/*	The name of the parent field.
SOURCE/*	Optionally, a field name which value is to be copied to the PARENT field before populating the subfields. Using this syntax would be equivalent to: PARENT = SOURCE; STAT = SUBFIELD (4, PARENT)

STAT is returned with one of the following codes:

STAT/I	1: OK -2: Wrong number of arguments -3: SOURCE field not same type or length as PARENT field -5: No subfields found for the parent field -(10+n): Subfield n not found in parent field

H.11 Array Processing Subroutines

This group of subroutines facilitate manipulation of the data elements of arrays.[12]

H.11.1 BINSRC - Binary Search in RMO Tables and Arrays

The Binary Search subroutine, BINSRC, performs an efficient lookup function on Record Maintenance (RMO) tables and/or arrays. Rather than having to program table or array searches for lookup values, you can conveniently achieve the same result with BINSRC.

TABLE Statements in an RMO (see Section 9.8 "TABLE Statement") allow you to address table file field names (or their synonyms) with an array subscript value corresponding to the record number in the table file. If there are NELE records in the file and, for example, 10 fields, the RMS can address 10 arrays by field name, where each array contains NELE elements.

You may also create and address local arrays as described in Section 9.5.1 "Creating Local Fields".

BINSRC requires the table arrays or local arrays be in sort on one or more search fields. Thus the table file must have been defined with one or more KEY or ASC fields, which become the search arrays, and have been sorted before the RMS is compiled. Local arrays used as search arrays must be preset or generated with data in proper increasing sort order.

Table search arrays or local search arrays for a given element number must have unique search data value combinations. For example, for tables defined with one or more key fields, each record in the table file must have a unique set of key values.

Should unique search array values not be present, BINSRC will find the specified search value combination, but which of the two or more corresponding sets of data values will be chosen is not determinate.

H.11.1.1 BINSRC Syntax

The syntax for performing a binary search of a sorted table is:

```
N = BINSRC(SARRAY1,...,SARRAYn,SVAL1,...SVALn,NELE)
```

SARRAYs/__	Up to five arrays are used as search arrays in order (SARRAY1, SARRAY2, etc.). Each of these arrays may have any data type.
SVALs/__	Up to five fields are used as search values. These values must be in the same order as the search arrays and must have the same field types as the corresponding search arrays.

12. Local arrays; RMO TABLE files (which are stored as local arrays in the RMO); and real fields that are referenced using array notation file can be manipulated using these subroutines.

NELE/I	Number of elements in the arrays furnished by the user. For table arrays NELE is the number of records in the table file. The user is responsible for supplying the correct value of NELE. If NELE is too small, only the first NELE array elements will be searched. If NELE is too large, BINSRC will search in some random place.
N/I	The subscript is the array or element number where BINSRC found the specified search value(s). A zero result means the search combination was not found. This may mean that the search arrays were not in sort. A -1 result indicates either a field type mismatch between SARRAY and SVAL fields or that NELE is not an integer (I) field. Since there must be an odd number of arguments for the subroutine, twice the number of search fields plus NELE, a -1 result could also indicate an erroneous even number of arguments.

H.11.1.2 BINSRC Example

The table, LOOKUP.TAB holds 1000 code combinations each with a dollar value. The Binary Search Method and a Program Method are shown here.

```
*   LOOKUP.TAB
*
MAS 1000
CODE1 A2 KEY1
CODE2 A2 KEY2
CODE3 A2 KEY3
DOLLARS D2
```

```
*   CODELOOK.RMS
*
*   Given Codes AA, BB & CC from DATA.MAS,
*   lookup AMOUNT in LOOKUP.TAB.
*   If no lookup record is found then set AMOUNT = 0.
FILE DATA.MAS
TABLE LOOKUP.TAB
LOCAL
NELE/I 1000
I/I
PROGRAM
*   Binary Search Method
*
I = BINSRC(CODE1,CODE2,CODE3,AA,BB,CC,NELE)
IF I GT 0 THEN AMOUNT = DOLLARS(I) ELSE AMOUNT = 0 END
*
.
.
*   Program Method
I = 0
LOOP: I = I + 1 ;
      IF AA EQ CODE1 AND BB EQ CODE2 AND CC EQ CODE3
         THEN AMOUNT = DOLLARS(I) ; GOTO DONE END ;
      IF I LT NELE THEN GOTO LOOP ELSE AMOUNT = 0 END
DONE: STOP
```

H.11.2 The SORT Subroutine

The SORT subroutine enables the rearrangement of several arrays in specified ascending or descending sort orders based on the values in one or more of the arrays.

In effect, SORT creates pseudo-records across several arrays. Record 1 consists of the first value of each array, record 2 consists of the second value of each array, record 3 the third, etc. These "records" are then sorted on the values in the arrays. SORT can sort each array in either ascending or descending order on an array by array basis.

H.11.2.1 SORT Syntax

The syntax of the SORT subroutine is:

```
STAT = SORT(DIREC,NVALS,AR1,AR2,...)
```

STAT/I	Returns the status of the SORT instruction. 1 means the call was successful -1 means that NVALS is larger than 6000 -2 means that DIREC contains over 10 A's and/or D's -3 means that DIREC contains other than A,D, or space -(10+N) means that there is too much data in Nth array
DIREC/An	An alphanumeric field containing the letters A and D, indicating ascending and descending, for each of the arrays on which the sort is based. The sequence of A's and D's in this field must be upper case and must be followed by a blank; for example, DIREC/A5 = 'AAAA '. The number of arrays on which the sort is based is determined by the number of A's and/or D's in DIREC. The maximum number of sort arrays is 10.
NVALS/I	Indicates the number of values in each array to be sorted. If any of the arrays has more than NVALS values, those values will not be sorted. If any of the arrays contain fewer than NVALS values the sort may fail, and the resulting arrays may be incorrect.
AR1,...	The field-names of the arrays to be sorted. Arrays of any field type occupy the third argument position on. The first array(s) named are used to form the sort key. For example, if DIREC contains four A's, the first four arrays are the basis for the sort. Any additional arrays are reorganized based on the sort sequence.

The internal limit of SORT is: (NVALS * 2) plus the total size (in words) of the data in the array with the largest sized values must not exceed 12000. E.g. If NVALS equals 100 and the arrays being sorted have field types X9999 and A70, then the A70 array has the largest size value (35 words). Since (100 * 2) + (100 * 35) equals 3700 the SORT will not exceed the limit of 12000 words.

H.11.2.2 SORT Example

To illustrate:

```
.
.
.
ARRAY1/A1(6)   'C' 'B' 'A' 'B' 'A' 'C'
ARRAY2/I(8)     1   2   3   4   5   6   7   8
ARRAY3/I(10)    4   2   3   2   4   2   3   2   4   2
```

```
DIREC/A2 'AA '
NVALS/I   6
STAT/I
.
.
.
STAT=SORT(DIREC,NVALS,ARRAY1,ARRAY2,ARRAY3)
```

After SORT has been invoked the arrays will contain:

```
ARRAY1/A1(6)  'A' 'A' 'B' 'B' 'C' 'C'
ARRAY2/I(8)    3   5   2   4   1   6   7   8
ARRAY3/I(10)   3   4   2   2   4   2   3   2   4   2
```

Note that the elements in the arrays beyond the sixth element remain intact.

H.11.3 ARSZ Subroutine

The ARSZ subroutine returns the size of an array (i.e. the number of elements it is dimensioned for, or the number of records in the TABLE file it was loaded from).[13]

```
SIZE = ARSZ(ARRAY)

SIZE/I    Size of array
ARRAY     Name of array
```

If ARRAY is a field that is not a local array, ARSZ will return a SIZE of 1.

H.11.4 ARINI Subroutine

The ARINI subroutine initializes an array by setting every element to the same value. If no VALUE argument is given, each element of the array is set to the null value (blank for An fields; zero for other fields). Unless there is an error, ARINI returns the size of the array.

```
SIZE = ARINI(ARRAY [,VALUE])

SIZE/I    Size of array
          -1: VALUE has different type than ARRAY
          -2: ARRAY is a TI or TX field
          -3: Incorrect number of arguments
ARRAY     Name of array
VALUE     Optional initial value (field or constant)
```

If ARRAY is a field that is not a local array, ARINI will initialize it and return a SIZE of 1.

13. RMO TABLE statements create local arrays at compilation, one for each field in the TABLE file. The dimension of these local arrays is the number of records in the TABLE file. The ARSZ subroutine is especially useful for determining the number of records loaded by a TABLE statement. See Section 9.8 "TABLE Statement" for a description of the RMO TABLE statement.

H.11.5 ARFND Subroutine

ARFND finds a value in an array by making a linear search of the array elements.[14] If VALUE is present more than once in ARRAY, ARFND returns the subscript of the first occurrence.

If ARFND is called with two arguments, the entire array is searched. With three arguments, ARFND begins searching at element START of ARRAY and continues to the end of the array. With four arguments, the search begins at element START and continues for NELE elements. If both START and NELE are given, and START is within the array but START + NELE - 1 is past the end of the array, ARFND stops searching at the end of the array: an out-of-bounds NELE value is not treated as an error.

```
N = ARFND(ARRAY,VALUE [,START [,NELE]])
```

N/I	>0: Subscript where VALUE was found in ARRAY 0: VALUE not found in ARRAY -1: VALUE has different type than ARRAY -2: ARRAY is a TI, TX or BLOB field -3: START or NELE is not an I field or an I constant, or its value is LE 0 -4: START is larger than size of ARRAY -5: Incorrect number of arguments
ARRAY	Name of array, or (if a scalar alpha local RMO field) the name of a field that contains the name of the array. The array (or array name) may also be the first field of a number of consecutive fields of the same data type in the virtual record's main file. The size (or dimension)of the array in this case is the number of consecutive fields of the same type (and length) that are specified in the DEF (the number of elements to consider can always be controlled using the START and NELE arguments).
VALUE	Value to find (field or constant)
START/I	Optional subscript at which to start searching (field or constant). If NELE is not given, the search continues to the end of the array.
NELE/I	Optional number of elements to search (field or constant). If NELE is used, START is required.

14. Because ARFND makes a linear search of the array the array elements do not have to be in any order. However, as a linear search through a large array could be time consuming, ARFND is intended for searching fairly small arrays. See the BINSRC subroutine, described in Appendix H.11.1 "BINSRC - Binary Search in RMO Tables and Arrays" for an efficient method of searching large arrays.

H.11.6 ARNONL Subroutine - Locate Non-Zero/Non-Blank Element

Use ARNONL to locate non-zero or non-blank elements in an array. E.g. in a financial application, you could use ARNONL to check if an account has any value other than zero in any of the twelve monthly accounting periods.

ARNONL syntax:[15]

```
STAT = ARNONL(ARRAY|AR_NAME,FIRST,LAST[,FLAG])
```

where

STAT/I	Returned with 0 if no non-zero (or non-blank) element was found, or subscript where a non-zero element was found, or: -1: Invalid data type (text or blob) -2: FIRST is < 1 or > LAST -3: LAST is > array size -4: Invalid number of arguments
ARRAY/x(n) or AR_NAME/An	Array of any data type, except TI, TXor BLOB. or Local scalar RMO field that contains the name of an array
FIRST/I	First element of array to check, starting with 1.
LAST/I	Last element of array to check.
FLAG/I	(Optional)Controls the direction of the search, and what to search for. If FLAG contains the value '1' the search will be conducted backwards (i.e. start with the last element and move towards element 1). If FLAGS contains the value '2' it will search for a NULL value instead of a non-NULL value. The First NULL value will be reported. These options can be combined by adding the FLAG values. FLAG = 3 will search for a NULL value from the end of the array towards the front.

After a non-zero element is found, the next non-zero element may be located by setting FIRST to STAT + 1, and calling ARNONL again.

If both the FROM and TO fields have a value of zero (0), FROM will be assigned the value 1, and TO will be assigned to the size (dimension) of the array.

Example:

```
FILE N.MAS
LOCAL
STAT/I
FIRST/I
LAST/I 12
   * Array has activity for last twelve months
```

15. Array element numbers used and returned by ARNONL start with one (1). When the array is a reference to a base field in the main file, the value returned must be referenced as ARRAY(STAT - 1) as the base of a field array is element 0 (see Section 8.7 “Arrays”).

```
WAR/D2(12) 0  0  0  20.00  0  0  123.00  0  0  0  0  0
D2/D2
PROGRAM
*
* Find most recent (last) non-zero value
*
FIRST = 1 ;
LOOP: STAT = ARNONL(WAR,FIRST,LAST) ;
   IF STAT NE 0 THEN ;
      D2 = WAR(STAT) ;
      FIRST = STAT + 1 ;
      IF FIRST LE LAST THEN GOTO LOOP END ;
      END
STOP
```

H.12 ASCII I/O Subroutines

The subroutines in this group provide access to ASCII "text" files.

H.12.1 ASCOPEN - Open ASCII File

ASCOPEN opens (read), creates (write), or opens for appending records (append) an ASCII file for further processing by the other ASCII I/O subroutines. Only one file may be open at a time.

```
STAT = ASCOPEN(WHICH,FILENAME)
```

where

STAT/I	>0: FNO for opened file -1: Could not open file -2: Invalid operation (WHICH not 'RD', 'WR', or 'AP') -3: FILENAME not An field
WHICH/A2	(field or constant) 'RD' to open file for reading 'WR' to create file for writing 'AP' to open or create file for appending records
FILENAME/ An	(field or constant) Name of file to be opened.

FILENAME may be provided via a File Open Dialog box, as described in Appendix H.12.1.1 “File Open Dialog Box”.

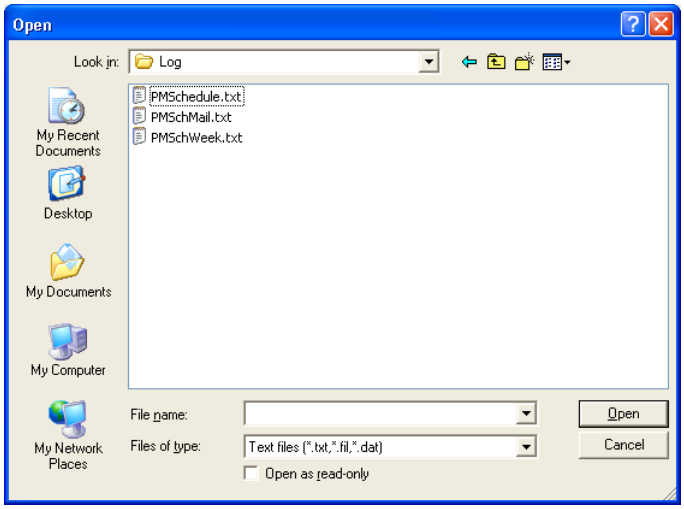
H.12.1.1 File Open Dialog Box

Certain RMO subroutines[16] support the ***File Open Dialog Box,*** which enables the user to browse to identify a target file for the subroutine call.

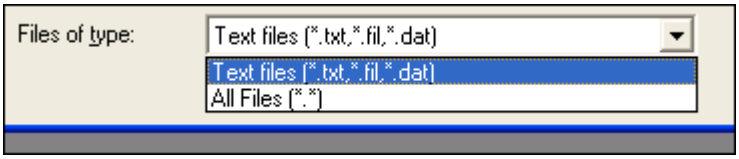
16. The following RMO subroutines support the File Open Dialog Box: ASCOPEN - Open ASCII File, CHECKFILE - Check Whether File Exists, BLOBIO - Access Binary Large Object (BLOB) Field , and VIEWTEXT: Display Text File in TRANS.

When calling these subroutines, set the argument for the target file name to '?'. This specifies that the file name will be supplied by selecting from a file open dialog box.

By default, the file open dialog box file displays the user's default folder, and uses a filter to display all files with extensions 'txt", "fil", and "dat'.



By default, the '"Files of type:" combo box also contains a 'All files' '*.*' choice.



If the argument for the target file is "?." (question mark followed by a period), the current working directory will be the starting directory for the dialog box[17].

The developer can customize the filters and the labels for the filters that will be used in the dialog box. If the "?" or "?." in the target file name field is followed by a "~" (tilde), the rest of the field can be used to specify a file selection filter using the following format:

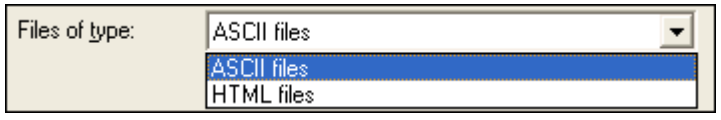
```
Display text~Filter~[Display text~Filter~...]~
```

where:

Display text is the label for the filter shown in the 'Files of type:' combobox. For example, "Text files (*.txt)". **Filter** is a wildcard filter (similar to what you would use for the DIR command) to select the files to be displayed. Any number of pairs of 'Display text' and 'Filter' may be given.

```
'?~ASCII files~*.txt;*.asc~HTML files~*.html;*.htm;*.htx~~'
```

shows, by default, **'*.txt'** and **'*.asc'** files (labelled "ASCII files"), but you may also select to show files with extension **'*.html'**, **'*.htm'** and **'*.htx'**.(labelled "HTML files")



17. The Current Working Directory may be set using the SETJPI subroutine (see Appendix H.14.12 "SETJPI - Setting/Changing Process Information").

H.12.2 ASCREAD - Read ASCII File

ASCREAD reads one line of text from the ASCII file opened by ASCOPEN and places the line the specified field or array.

```
STAT = ASCREAD(FNO,BUF,LEN)
```

where

STAT/I	>=0: Number of characters read -1: EOF -2: No file open for read -3: BUF not An field -4: LEN >254
FNO/I	File number to read.
BUF/An	(field) Buffer to receive one line of text to be read. BUF may be an array of alpha fields (e.g. BUF/A80(4)), or a number of adjacent G$ fields. If more than one field is to be used, LEN must state the total length of the buffer to use.
LEN/I	The length of BUF in bytes. If BUF is only one field, LEN may be given as zero, and the actual length of the BUF field will be used. If BUF is an array, or a series of G$ fields, LEN must state the total length of the buffer.

H.12.3 ASCWRITE - Write ASCII File

ASCWRITE writes the contents of the specified field or array into the ASCII file created/opened by ASCOPEN. The syntax is

```
STAT = ASCWRITE(FNO,BUF,LEN [,OPT]
```

where:

STAT/I	1: OK -2: File not open for write -3: BUF not An field -4: LEN > 254 -5: Wrong number of arguments; should be 3 or 4.
FNO/I	File number to write.
BUF/An	(field) Buffer with text to write. May be an array of alpha fields (e.g. BUF/A80(4)), or a number of adjacent G$fields. If more than one field is to be used, LEN must state the total length of the buffer to use.

LEN/I	The length of BUF in bytes. If BUF is only one field, LEN may be given as zero, and the actual length of the BUF field will be used. If BUF is an array, or a series of G$ fields, LEN must state the total length of the buffer.
OPT/I	Optional. (Field or constant.) 1: Retain trailing blanks 2: Translate characters using TRANS_ENV DMAP 4: Translate characters using loaded translation map. 8: Allow more than 254 characters in output.

To combine options, add the values for the options together (for example, if you want both option 1 and 2, set OPT = 3.)

ASCWRITE translates characters using a DMAP translation table from the TRANS_ENV file when OPT is set to 2. If OPT is set to 4, ASCWRITE will load and use a DMAP translation table that only affects ASCWRITE operations, specifically loaded for that purpose by a special ASCOPEN call[18].

NOTE

There is no need to reset the table if you do not want ASCWRITE() to perform any translations. Just ensure that '4' is not in OPTION.

18. When ASCOPEN() is called with open mode set to 'LT' (Load Translation table) and a file name is included that is the path to a file containing DMAP syntax, the system loads that file as the translation table for ASCWRITE() only. Subsequent calls to ASCWRITE() with OPTION 4 use this table. Once loaded, the table stays in effect for the whole TRANS session or until another 'LT' ASCOPEN() call replaces it.
The ASCOPEN() 'LT' call only processes lines starting with "dmap:". So the file you give it can be a regular TRANS$ENV file with DMAP entries, or a file with only DMAP entries made specifically for this purpose. There is no error checking with the exception of a 'file not found' error.

H.12.4 ASCCLOSE - Close ASCII File

Use ASCCLOSE to close the file created/opened by ASCOPEN.

```
STAT = ASCCLOSE(FNO)
```

where

STAT/I	1: OK -1: File not open
FNO/I	File number to close.

H.12.5 DELFILE - Delete File

DELFILE deletes the specified file.

```
STAT = DELFILE(FILENAME)
```

where

STAT/I	1: OK -1: File not deleted
FILENAME/ An	(field or constant) Name of file to be deleted.

H.13 Subroutines that Modify or Control TRANS

The subroutines in this group are used to modify or control TRANS behavior "on the fly".

H.13.1 AUTOBR: Automatic Branch Control

The AUTOBR subroutine can be used to dynamically control access to screens, identified by their branch codes, based on logic in the RMO. Manual branching[19] to a screen or set of screens can be activated or deactivated at any time with AUTOBR. AUTOBR can set branch codes as manual or automatic-only, whether or not those branch codes appear in the screen where AUTOBR is called. The setting of a particular branch code remains in effect as long as the user is in TRANS, unless another call to AUTOBR changes it. Therefore, AUTOBR can be used in a login screen or a menu screen to set branch choices in other screens which the user may visit.

19. AUTOBR only controls branches which are designated in the TRS as automatic-only (with '%%' preceding the branch message text). Normally, these branches are hidden from the TRANS user and are not available for manual branching.

When AUTOBR is called with ACTION = 0 and an array of branch codes, then, when the branch codes in the array are encountered, they are treated as manual branches (the '%%' is removed from the displayed branch message).

When AUTOBR is called with ACTION = 1 and an array of branch codes, then, when these branch codes are encountered with the '%%' designation in the TRS, they are treated as automatic-only branches, as they normally would be.

If AUTOBR is called with ACTION = 2, then all '%%' branches are treated as automatic only (that is, ACTION = 2 restores the normal branching action for all subsequent '%%' branches).

AUTOBR branch settings take effect immediately (next time TAB is pressed). The branch settings remain in effect as long as the user is in TRANS, unless they are changed by another AUTOBR call. The setting of one or more branch codes can be changed at any time, in any order. For example, a menu screen might set branches A, C, and E as manual; then a subsequent screen might change C and E to automatic only; later C could be set back to manual and A to automatic only; etc. AUTOBR can be used more than once in the same screen: for example, a screen could set branch code C to manual and then later set it back to automatic only. When TAB is pressed, TRANS treats branch C as manual or automatic only, according to the setting established by the last call to AUTOBR which set the action for branch code C.

Since the branch codes supplied to AUTOBR do not need to appear in the screen where AUTOBR is called, TRANS cannot verify that they are branch codes for '%%' branches. A branch code supplied to AUTOBR has no effect until the user enters a screen where the branch code is used for a '%%' branch.

At any given time, a maximum of 100 '%%' branch codes can be designated as manual branches using AUTOBR (AUTOBR returns an error status, and does not change the branch settings, after this limit is reached). This limit is of no concern unless the application contains over 100 different branch codes. AUTOBR prevents duplication: for example, if branch code A is set to manual twice in a row, it only counts once against the limit of 100 branch codes set to manual.

Syntax:

```
STAT = AUTOBR(ACTION,BRANCHES)
```

STAT/I	Execution Status 1: OK: Branch actions were set as requested. -1: Value of ACTION is not valid. -2: AUTOBR cannot set any more branch codes as manual branches. Currently, 100 different branch codes are set to manual; AUTOBR has been asked to set more and it cannot. -3: The BRANCHES array contains over 100 branch codes, or it is not terminated with a blank. This is an error in the RMS which must be fixed. Future branches and calls to AUTOBR may not function as expected.
ACTION/I	Action Code (field or constant) 0: Make the branches in the BRANCHES array manual. 1: Make the branches in the BRANCHES array automatic-only. 2: Make all branches function as designated in the TRS (manual if no '%%', automatic-only if '%%').
BRANCHES/ A2(n)	Branch Code Array (field) This array must be of type A2 and must not contain more than 100 branch codes. Branch codes can be in any order and must be uppercase. The end of the BRANCHES array is indicated by a blank array element, which must be present. For example, BR/A2(3) 'A' 'CD' ' '. If ACTION = 2, this argument can be any An field: it must be present, but its contents are not used.

H.13.2 Bitmap - Placing Bitmaps in Trans

The BITMAP subroutine places bitmaps (icons etc.) in the TRANS client area. The general syntax is:

```
STAT = BITMAP(FUNC,Y,X,FLAG,IDENT [,CY [,CX]])
```

where

STAT/I	1: OK -1: Unknown operation -2: Unknown FLAG value -3: Invalid number of arguments -4: Invalid predefined Icon -5: Argument 5 has wrong type -6: Argument 6 has wrong type -7: Argument 7 has wrong type -9: Invalid icon identifier -10: Unable to identify icon at this location
FUNC/I	Identifies which function to perform. Currently, the following function is defined: 0: Paint Bitmap 1: Paint Icon 2: Paint Jpeg (.JPG) file (a Jpeg file is converted to a bitmap before being painted) 99: Remove bitmap from position Y, X
Y/I	Line number for upper left corner of icon or bitmap
X/I	Column number for upper left corner of icon or bitmap

FLAG/I — Identifies how the following arguments are interpreted:

4096: Paint image on top of label text.

If FUNC = 0: Paint Bitmap:

32: CY and CX are given in pixels.

If FUNC = 2: Paint Jpeg image:

2, 4, 8: Scaling factor to apply when converting the Jpeg file to a bitmap. Image will be scaled to 1/2, 1/4, or 1/8 of original size (applies both horizontally and vertically).

32: CY and CX are given as pixels.

If FUNC = 1: Paint Icon:

1: Next argument is an I field identifying which built-in icon to paint. The following are available:

IDENT/I

1: Application
2: Information
3: Question
4: Exclamation/Warning
5: Error/Stop
6: Windows Logo
10: ADMINS logo
11: Post It
12: Happy face
13: Sad face
20: Excel (32 pixels)
21: Excel (16 pixels)
22: Word (32 pixels)
23: Word (16 pixels)
24:Calculator
0: Clear out previous logo

IDENT/An — Pathname of Icon or Bitmap or Jpeg file to load and paint.

Or, in case FUNC = 1 and FLAG = 1:

IDENT/I — Predefined Icon number to paint (as described above).

CY/I — Used for bitmaps and icons.

Optional vertical size of bitmap.

For icons:

0 = Use actual size (same as no CY argument)
1 = Use system default (normally 32 pixels)
2 = Use 16 pixels
4 = Use 32 pixels

CX/I — Used for bitmaps only. Optional horizontal size.

If no size is given (CY and CX not present, or contains zero), the actual size of the bitmap is used.

The size can be given as number of lines (CY) and number of columns (CX), which is hard to use without distorting the picture (since normally a line is longer than a column is wide). By setting FLAG to 32 CY and CX can be given in pixels, which gives you full control of the size of the bitmap.

H.13.3 Button - Creating and Modifying Buttons in TRANS

The BUTTON subroutine creates new buttons or modifies the appearance and behavior of buttons in TRANS. The syntax of the BUTTON subroutine is:

```
STAT = BUTTON(BTNNAME,OP,VALUE[,OP2,VALUE2...])
```

where:

STAT/I	1: OK -1: BTNNAME is not type An (alphanumeric) -2: BTNNAME button not found -3: Invalid OP code -4: Invalid state code -5: Cannot press a grayed button -6: Invalid number of arguments -10n: Argument n has wrong type -20n: Invalid action code in argument n -30n: Invalid video code in argument n -40n: Invalid icon number in argument n -50n: Invalid bitmap path in argument n
BTNNAME/ An	Field or constant containing the name of the button to modify.
OP/I	Operation to perform: 1: Change button label 2: Change action codes 3: Change video attributes 4: Change Down video attributes 5: Change Grey video attributes 6: Set state 7: Add/Change hover text 8: Specify/Change font number 9: Text and bitmap justification 10: Set Background Color 11: Set Foreground Color (black is default) 12: Set Background Color for down state 13: Set Foreground Color for down state 14: Set Background Color for grayed state 15: Set Foreground Color for grayed state 20: Display icon 22: Display bitmap 25: Size of bitmap in pixels 99: Delete button A negative OP code restores the original value (e.g. OP = -1 restores the original button label). (There is no -99 OP code). The following codes may be used to create new buttons (must be 1st code in list): 258: Create a new Pushbutton 259: Create a new Labelbutton

VAL/* Value field type and content depend on the preceding OP code:

OP = 1: An with new label

OP = 2: An with new action code(s). See Section 5.5.21 "Button Objects in TRANS" for rules on how to specify action codes.

OP = 3, 4 and 5: An with video codes. See Section 5.5.21 "Button Objects in TRANS" for rules on how to specify action video codes.

OP = 6: Integer. Allowed values are:

0: Normal state

1: Simulate button press

2: Greyed state (no action is taken while in this state).

OP = 7: An with the hover text to be displayed.

OP = 8: Integer with font number (defined in TRANS_ENV file).

OP = 9: Integer. Text and bitmap justification codes.

1 = Left justify text

2 = Right justify the text

16 = Center the image

32 = Right justify the image

OP=10,11,12,13,14,15: An I(3) array with RGB codes, or An field with color name.

In order to change the color of a button it must be created with a color attribute. E.g.

,10,'gray/A4'

creates a button that looks like a button created without a color attribute, but has the properties of being able to change its color in a later call to the BUTTON subroutine.

OP = 20: An Integer field containing the id number of one of the "known" icons (see the BITMAP subroutine for a list of known icons), or an An field containing the pathname of an icon to display.

OP = 22: An An field containing the pathname of a bitmap to display.

OP = 25: An I(2) array containing the width and the height of the bitmap in pixels. If not present the size of the bitmap is assumed to be 32 pixels (both width and height).

OP = 99: Any field (dummy).

OP = 258 and 259: An integer array, where:

I(1) = X value (column number)

I(2) = Y value (line number)

I(3) = Number of columns

I(4) = Number of lines

All OP codes require a VAL argument, although the negative OP codes and OP code 99 does not care about its type or value. A good practice is to specify those value arguments as 0 (zero). For example:

```
STAT = BUTTON('EXIT',-1,0)
```

restores the original label for the button named EXIT.

All arguments to the BUTTON subroutine may be given as fields or constants. The VAL arguments must be include the field type if they are given as alphanumeric constants, e.g.

```
STAT = BUTTON(BTNNAME,2,'%brnc 1/A7')
```

H.13.4 TEXTOUT - Display Text in TRANS using any Font or Color

The TEXTOUT subroutine can be used to place text of any font and color in the client window. The syntax is:

```
STAT = TEXTOUT(OPTION,Y,X,TEXT [,SIZE [,LEN [,FONT [,FG [,BG]]]]])
```

where:

OPTION/I	1 **Bold** 2 Underline 4 Italic These options may be combined by adding them together. 8 Always use the Main screen as base for calculating X/Y offsets for the text. 1024 Delete text previously output at the location specified. NOTE: if the text was created with option 8 you'll have to add 8 to 1024, setting option to 1032 before calling TEXTOUT. To **delete all text previously placed by TEXTOUT** set both **X** and **Y** arguments to **-1** (OPTION value not evaluated in this case)
Y/I	Line number at which to start the display.
X/I	Column number at which to start the display.
TEXT/An	Text to display. Leading and trailing '^' characters may be used to align the text. These four arguments are the only mandatory arguments. If no further arguments are presetn, the default font and size for label text are used.
SIZE/I	Point size to use.
LEN/I	If non-zero, length of text. If LEN is greater than the number of non-blank characters in TEXT, the text is padded with spaces.
FONT/An	Name of font family to use (e.g. 'Courier New')
FG/An or / I(3)	Foreground color to use.

BG/An or / I(3)	Background color to use. Colors may be given as a color name as specified in TRANS Built-in Colors, or as an RGB value in an integer array.
STAT/I	Return value: 1 = OK -1 = Invalid number of arguments (< 4 or > 9). -2 = Invalid Y or X value -3 = TEXT has invalid format -4 = FOREGROUND has invalid type (not An or I) -5 = FOREGROUND has invalid value -6 = BACKGROUND has invalid type (not An or I) -7 = BACKGROUND has invalid value

H.13.5 PRTSCR - Printing the Client Area of a TRANS Screen

The PRTSCR subroutine prints the client area of a TRANS screen. PRTSCR first copies the selected area of the screen to the clipboard, and then optionally invokes TedRE to print the content of the clipboard. The general syntax is:
STAT = PRTSCR(FLAG [,Y,X,NLINES,NCOLS])

where:

FLAG/I	Bitmap flag to modify the behavior of PRTSCR. The following flags are defined: 1: Copy to clipboard only (do not print). 2: Measurements are in pixels 4: Use whole window area, measurements in pixels 8: Print using landscape mode If FLAG is the only argument to PRTSCR, the whole client area of the screen will be copied and printed. To copy and print a selected area on the screen the next four arguments must be present:
Y/I	Line number for upper left corner to be copied. If measurements are in pixels (FLAG equals 2 or 4), a negative Y means measure Y pixels up from the bottom.
X/I	Column number at which to start.
NLINES/I	Number of lines to copy.
NCOLS/I	Number of columns to copy. If measurements are in pixels (FLAG equals 2 or 4), X equals -1 refers to the right edge of the area (client or window).

H.13.6 DISPFLDS: Modify Field Display List in TRANS

Use the DISPFLDS subroutine in RMOs running with TRANS to modify the list of fields which are displayed in a screen. DISPFLDS can be called at any time from the RMO.

DISPFLDS processes a list of field names in an local RMO array, and modifies the list of fields to be displayed in one of the following two ways:

1. the fields on the list should be the only fields displayed, or
2. the fields on the list should not be displayed.

If a field is displayed, DISPFLDS has no effect on whether it is editable. (to change the edit status of fields, use the EDFLDS subroutine, described in Appendix H.13.7 "EDFLDS - Modify List of Editable Fields in TRANS"). Of course, if a field is not displayed, it is not editable either. Therefore, when DISPFLDS blocks the display of a field which is declared in the TRS as editable, it also prevents the cursor from going to the field.

The syntax:

```
STAT = DISPFLDS(ACTION,FIELDS)
```

STAT/I	1: Successful 0: Nothing done, DISPFLDS was called after first pair of BEGREC calls in screen. -N: The Nth element of FIELDS does not match any field in the screen. If N is larger than the number of elements in FIELDS, the problem is that the last element of fields is not a blank element. -1000: Over 1000 elements in FIELDS, or more elements than there are fields in the screen.
ACTION/I	(Field or constant) 2: Allows a subset of fields, that have been turned off by a previous call of -1, to be selectively turned back on. 1: Display only the fields named in FIELDS -1: Do not display the fields named in FIELDS 0: Display all fields as specified in TRS 2: Redisplay the fields named in FIELDS (useful after -1 used to prevent display of a group of fields)
FIELDS/An(n)	(Array) Contains a list of field names, which can be abbreviated. These fields should be specified in the fields section of the TRS. The last element in the array MUST BE A BLANK. When ACTION is zero, the contents of this array are not used, but the argument must be present.

H.13.6.1 DISPFLDS - Example

In the following example, the field G$USER_ACC is checked to determine whether the current user should be able to view the SALARY field. If the user is not authorized, DISPFLDS is called to block display of SALARY.

```
FILE PAYROLL:EMP.MAS
*
S$S/A6
M$M/A2
G$USER_ACC/A1
STAT/I
DFACT/I
DFLIST/A10(2) 'SALARY' ' '
XCALL/I 0
*
PROGRAM
*
IF XCALL EQ 1 THEN GOTO ALLCALLS END
FIRSTCALL: XCALL = 1
IF G$USER_ACC EQ 'S' THEN GOTO ALLCALLS ELSE ;
    DFACT = -1 ; STAT = DISPFLDS(DFACT,DFLIST) ;
    END
ALLCALLS: ;
...
```

H.13.7 EDFLDS - Modify List of Editable Fields in TRANS

Use the EDFLDS subroutine in RMOs running with TRANS to modify the cursor order of the screen (the list of editable fields).

EDFLDS gives developers a straightforward way to restrict cursor movement in TRANS applications. Some of the situations where you might use EDFLDS are:

1. Field level security, where, based on logic in the RMO, the user is not allowed to change certain fields;
2. "Sub-screens", where one screen contains several logical sections, and you want to restrict the cursor to one subscreen at a time.

EDFLDS lets you specify a list of field names that should be the only editable fields; or specify a list of field names that should not be editable. EDFLDS can also restore the list of editable fields to that specified in the TRS.

EDFLDS can be used at any RMO call, and the new cursor order takes effect immediately. All the field names used with EDFLDS must be specified as editable fields in the TRS; otherwise EDFLDS will have no effect on them (EDFLDS can change a normally editable field to display only, but not the reverse).

Here, "editable" means that the cursor goes to a field. EDFLDS does not block changes to fields by the RMO; it just restricts where the cursor can go. **EDFLDS overrides ALLOW and C$C*.** If a field is ALLOWed or is specified in C$C, but EDFLDS has set that field as display only, then the cursor won't go to it. EDFLDS does NOT override ADM$READONLY.[20]

After EDFLDS has modified the normal cursor order, "query" field selection mode in TRANS[21] displays and goes to only those fields specified by EDFLDS as editable.

EDFLDS can make key fields display only. If KEY1 is made display-only by EDFLDS, then the HOME key puts the cursor at the first non-key field which can be edited. If KEY2 and/or KEY3 are display only, then the subkey keystrokes [(the KEY2 keystroke) and] (the KEY3 keystroke) go either to KEY1, if it is editable, or to the first editable non-key field if KEY1 is not editable.

---

20. See Section 5.5.14 “ALLOW statement” for details on ALLOW, see Section 16.3 “Cursor Control: C$C and C$MULREC” for details on C$C, and see Section 6.4 “Entering or Changing Fields” for details on ADM$READONLY.
21. see Section 5.3.1.4 “TABBING or QUERY: Field Selection Mode”

Normally, all key fields should be editable, or none should be. If some key fields are editable and others are not, the user can reach the highest editable key using the HOME key or a subkey key (KEY2 or KEY3). If a value is entered for that key, then the cursor will go to all lower keys, regardless of the EDFLDS setting.

Syntax:

```
STAT = EDFLDS(ACTION,FIELDS)
```

STAT/I	1: Successful -N: The Nth element of FIELDS does not match any field in the screen. If N is larger than the number of elements in FIELDS, the problem is that you neglected to end FIELDS with a blank element. -1000: Over 1000 elements in FIELDS, or more elements than there are fields in the screen.
ACTION/I	(Field or constant) 1: Make fields in FIELDS array the only editable fields -1: Make fields in FIELDS display only 0: Return to normal editable field list specified in TRS. -2: Make fields in FIELDS display only and blocks the cursor from going to text and display fields with Lookup.
FIELDS/An(n)	(Array) Contains a list of field names, which can be abbreviated, provided they uniquely identify the fields in question. These fields should be specified as editable in the TRS (this is not checked). The last element in the array MUST BE A BLANK. When ACTION is zero, the contents of this array are not used, but the argument must be present.

H.13.7.1 EDFLDS - Example

In the following example, the field Z_DATE is set to be display only via the EDFLDS subroutine (at every RMO call). When M$M has a certain specific value, EDFLDS is used again to reset Z_DATE back to editable.

```
FILE IS_DATA:ISC100.MAS
*
S$S/A6
M$M/A2
C$C/A6
.
.
.
STAT/I
EDFACT/I
EDFLIST/A10(2) 'Z_DATE' ' '
.
.
.
PROGRAM
*
* Use EDFLDS to set the Z_DATE field
* display only because its only intended
* for use in specific circumstances
*
EDFACT = -1 ; STAT = EDFLDS(EDFACT,EDFLIST)
```

```
.
.
.
* When M$M has the value 'HS' set the Z_DATE
* field editable, and send the cursor there.
*
HS: ;
IF M$M NE 'HS' THEN GOTO UX END ;
 EDFACT = 0 ; STAT = EDFLDS(EDFACT,EDFLIST) ;
 C$C = 'Z_DATE' ;
.
.
.
```

H.13.8 GBLSTORE - Access TRANS Global Area on Disk

The GBLSTORE subroutine allows an RMO running with TRANS to write the TRANS global area (the "G$" fields) to a disk file and later read it back into TRANS.

GBLSTORE is useful, for example, in an application where a user performs some operations in TRANS, then leaves TRANS and executes some other images, and then returns to TRANS. GBLSTORE can write out the user's global fields before leaving TRANS, and read them back in when the user returns to TRANS.

The GBLSTORE syntax is as follows:

```
STAT = GBLSTORE(OP,USERID)
```

OP/A2	(field or constant) 'W' to write global area to file 'R' to read global area from file 'RD' same as 'R', but delete file after reading 'D' just delete the file
USERID/An	(field or constant) Unique user ID string. If blank, the translation of ADM$TERM is used.
STAT/I	Status: 1: OK -1: OP code is not valid -2: File cannot be opened -3: ADM$TERM not defined

GBLSTORE stores global data for each user in a small non-ADMINS binary file. The names of these files have the form ADM$GBL:TRANS_<USERID>.GBL. The logical name ADM$GBL must be defined in order to use GBLSTORE. ADM$GBL points to the directory where GBLSTORE will look for global data files. Users must be able to read and write to the ADM$GBL directory. GBLSTORE opens (or creates) the global data file only when necessary; and, once it has been opened, file remains open until the user leaves TRANS or one of the delete options (OP = 'D' or OP = 'RD') is used.

The user's global file can be written and read at any time by application screens. Normally, it should only be written immediately before exiting from TRANS, and read immediately after coming back into TRANS.

OP = 'RD' deletes the user's global file immediately after reading it. The purpose of this option is to avoid the proliferation of user global files. If the application does not have user ID's of its own, GBLSTORE uses ADM$TERM for the user id in the global file name. But, depending on how ADM$TERM is assigned, it may have

unpredictable values (e.g., if it is based on an LTAxxx terminal number, which increments whenever anyone logs in). Alternatively, the 'D' option can be used to delete the global file when the user leaves an application; or the user's global file can be deleted in a DCL procedure when the user logs off.

H.13.9 NOEK - Set TRANS to Read Next Field With No Echo

It is possible to instruct TRANS to read a field with no echo, which may be used to input a password on a logon screen. This is done by letting the RMO behind the screen call the subroutine NOEK, which instructs TRANS not to echo the next field typed in. (Be aware of the fact that if the noecho field is defined as an ER field, the refresh mechanism will re-display the contents of the field after it is typed in. The RMO should therefore move the information into a local field right after it is typed, and blank out the noecho field.)

H.14.6.1NOEK Syntax

```
NULL = NOEK(NULL)
```

NULL/I	Required for syntax purposes only.

H.13.9.1 NOEK Example

NOEK would typically be used to input a user password which should only be known to the user and therefore should not be displayed on the screen. Given the following lines in a screen instruction file,

```
...
ER USER/X9999
ER PWD/A6
...
```

then an RMO behind the screen might contain the following:

```
...
S$S/A6
USER/X9999
PWD/A6
PASS/A6
NULL/I
PROGRAM
IF S$S EQ 'USER' THEN NULL = NOEK(NULL) END
IF S$S EQ 'PWD' THEN PASS = PWD ; PWD = ' ' END
...
```

When the field "USER" is entered TRANS will not echo the next field entered which would be "PWD". In addition, when the field "PWD" is entered it is saved and then "blanked out" so it will not be refreshed on the screen.

H.13.10 PAUSE - Create a Pause in TRANS

It might be useful to have TRANS pause at times before continuing to perform its current activity. For example, re-displaying the records in a multi-record screen every thirty seconds. One could make TRANS pause by executing a loop in the RMO behind the screen or by sending TRANS through a set of automatic branches. Either method consumes significant CPU or I/O resources, and would seriously degrade response on the other terminals.

There is a PAUSE subroutine, callable from an RMO running behind the screen, to allow TRANS to pause with no overhead.

H.13.10.1 PAUSE Syntax

```
NULL = PAUSE(NTICKS,UNITS)
```

NULL/I	Required for syntax purposes only.
NTICKS/I	Number of UNITS to pause.
UNITS/I	Unit of time measurement. 1 signifies ticks (hundreths of seconds) 2 signifies seconds 3 signifies minutes 4 signifies hours

H.13.10.2 PAUSE Example

Given the following fields and values,

```
NULL/I
NTICKS/I 2
UNITS/I 3
```

then

```
NULL = PAUSE(NTICKS,UNITS)
```

included in an RMO behind a screen would cause TRANS to "pause" for two minutes.

H.13.11 POLYDRAW - Draw Polygons or Connected Lines in TRANS

The POLYDRAW subroutine is used to draw any polygon or connected lines in the client area of TRANS. The syntax is

```
STAT = POLYDRAW(NPOINTS,Y,X,FLAG)
```

where

NPOINTS/I	The number of given Y/X coordinates.
Y/I(n) X/I(n)	Two arrays that contains pairs of Y and X coordinates for the corners of the polygon to be drawn. A minimum of two pairs of coordinates must be given (NPOINTS >= 2). Y and X must be given as two local arrays in the RMO.
FLAGS/I	Flags modifying the behavior of POLYDRAW. Currently the only FLAG defined is -1, which is used to remove a polygon from the current screen (polygons are automatically removed when branching to a new screen). If FLAG = -1, the ID of the polygon to remove is in NPOINTS.
STAT/I	>0: ID of polygon being drawn. 0: Unable to draw -1: Too many polygons (max 200 in one screen) -2: Must specify at least 2 points -3: Y must be of type I -4: The dimension of the Y array is less than NPOINTS. -5: X must be of type I -6: The dimension of the X array is less than NPOINTS.

H.13.12 POPUP - Displaying a Popup Menu in the Screen

The POPUP subroutine displays and allows the user to utilize a popup menu in the screen. The syntax is:

```
STAT = POPUP(Y,X,NITEMS,IDS,FLAGS,STRINGS)
```

where:

Y/I	(field or constant) Line number of upper left corner (may be 0).
X/I	(field or constant) Column number of upper left corner (may be 0).
NITEMS/I	(field or constant) Number of items in popup menu (including separators).
IDS/I(n)	(field) An array of ids where n is at least NITEMS. The id number is what is returned in STAT if this item is selected. The id of separators and next level popup menus (see FLAGS) are ignored, but must be present as a placeholder.
FLAGS/I(n)	An array of modifying flags, one for each item. The following flags are supported: 1: Gray the item 2: Disable the item (not grayed) 16: Next level popup menu 2048: Item is a separator All other items are strings to be displayed.
STRINGS/ Am(n)	(field) The string to display for each item. Separators do not have strings to display, but an empty placeholder must be provided. If the menu item is identified as a next level popup menu (FLAG=16) the next member of the IDS, FLAGS and STRINGS arrays are used to identify the fields used for these fields for the next level popup menu. IDS should contain 0, FLAGS should be set to number of items in the next level menu, and STRINGS should contain the field names of the arrays used for IDS, FLAGS and STRINGS for the next level menu, e.g. STRINGS(n) = `'ID2 FLAG2 STRING2'` means that the array ID2/I(n) is used to store Ids, FLAG2/I(n) is used for flags, and STRING2/Am(n) is used for strings for the next level menu.

STAT/I	Returned with the ID of the item selected (0 if no item is selected, so do not use 0 (zero) as an item id), or one of the following error codes: -1: IDS is not type I. -2: The dimension of IDS is not at least NITEMS. -3: FLAGS is not type I. -4: The dimension of FLAGS is not at least NITEMS. -5: STRINGS is not type A. -6: The dimension of STRINGS is not at least NITEMS. -7: Syntax error in sub-menu specification -8: Sub-menu ID field not found -9: Sub-menu ID field is not type I -10: Submenu ID array not large enough -11: Sub-menu FLAG field not found -12: Sub-menu FLAG field not of type I -13: Sub-menu FLAG array not large enough -14: Sub-menu STRING field not found -15: Sub-menu STRING field not of type A -16: Sub-menu STRING array not large enough

H.13.12.1 POPUP Example

This RMS code shows how to set up a 3-level cascading pop-up menu using the POPUP subroutine:

```
file RMOSUB:n.mas
local
m$m/a2
s$s/a6
stat/i
y/i 3
x/i 6
items/i 9
*
*first level popup menu
*
ids/i(10)   0 0    0  0 0  0 0  0 0
flags/i(10) 2 2 2048 16 4 16 7 16 5
strings/a24(10) 'Search People File'
                'Choose Method'
                ' '
                'Name Searches'
                'xid xflag xstring'
                'Place Searches'
                'yid yflag ystring'
                'Date Searches'
                'zid zflag zstring'
*
* second level 1 (cascaded)
*
xid/i(10)   11 12 13 14
xflag/i(10)  0  0  0  0
xstring/a28(10)  'Last Name'
                 'Last Name contains string'
                 'Last Name, First Name'
                 'Last Name sounds like'
*
* second level 2 (cascaded)
*
yid/i(10)   21 22 23 24 25   0  0
yflag/i(10)  0  0  0  0  0  16 2
```

```
ystring/a32(10)  'Zip code (last known)'
                 'Zip code (all)'
                 'City/Town (last known)'
                 'City/Town (all)'
                 'City/Town and Street Name (all)'
                 'Phone Number Searches'
                 'nid nflag nstring'
*
* second level 3 (cascaded)
*
zid/i(10)   31 32 33 34
zflag/i(10)  0  0  0  0
zstring/a24(10)  'Date of incident'
                 'Date of Birth'
                 'Date of Residence'
                 'Date of Official Record'
*
* third level (cascaded)
*
nid/i(10)   41 42
nflag/i(10)  0  0
nstring/a24(10)  'Area code'
                 'Area code, Exchange'
b$b/a2
program
if m$m eq '88' then ;
   stat = popup(y,x,items,ids,flags,strings) ;
   if stat eq 42 then b$b = 'X' end ;
   end
stop
```

H.13.13 READBR Subroutine: Read-Only Branch Access

The READBR subroutine can be used to turn read-only access[22] on and off for screens, identified by their branch-codes, prior to branching into them. Read-only access can be turned on or off at any time before a screen is branched to.

The branch codes acted on by READBR need not exist in the TRO that calls it. Thus, READBR can be called once, for example in a menu screen, to set up branch codes for a whole family of other screens. The setting of a particular branch code as read-only remains in effect as long as the user is in TRANS, unless another call to READBR changes it.

By calling READBR with an action code and an array of branch codes, branch codes can be added to or removed from the list of read-only branches at any time, in any order.

To set certain branch codes as read-only, call READBR with ACTION = 0 and an array containing the branch codes. When the user branches using these branch codes, the target screen will be in read-only mode. To restore one or more branch codes to "normal" (read/write) mode, call READBR with ACTION = 1 and an array containing the branch codes to be restored. Calling READBR with ACTION = 2 restores all branches to read-write mode without naming the branch codes.

READBR settings take effect at the next branch. The TRANS return branching features (CTRL/R and R$R) "remember" screens which were read-only when entered; if you return-branch to these screens, they remain read-only.[23]

---

22. as if ADM$READONLY were set (see Section 6.4 "Entering or Changing Fields").
23. If you branch to them normally (i.e. not via CTRL/R or R$R) they will be in whatever access mode is currently in effect for that screen's branch code.

Since the branch codes supplied to READBR do not need to appear in the BRANCHES paragraph of the screen where READBR is called, READBR cannot verify that the branch codes it is given exist. Nonexistent branch codes have no effect.

At any time, a maximum of 100 branch codes can be designated as read-only branches using READBR (READBR returns an error status, and does not change the branch settings, after this limit is reached). This limit is of no concern unless the application contains over 100 different branch codes. READBR prevents duplication: for example, if branch code A is set to read-only twice in a row, it only counts once against the limit of 100 branch codes.

If 'Y' is assigned to the logical name ADM$READONLY, all screens are read-only. READBR cannot override ADM$READONLY.

READBR Syntax:

```
STAT = READBR(ACTION,BRANCHES)
```

STAT/I	Execution Status 1: OK: Branch actions were set as requested. -1: Value of ACTION is not valid. -2: READBR cannot set any more branch codes as read-only branches. 100 different branch codes have already been set. -3: The BRANCHES array contains over 100 branch codes, or it is not terminated with a blank. This is an error in the RMS which must be fixed.
ACTION/I	Action Code (field or constant) 0: Make the branches in the BRANCHES array read-only. 1: Make the branches in the BRANCHES array normal 2: Make all branches function normally
BRANCHES/ A2(n)	Branch Code Array (field) This array must be of type A2 and must not contain more than 100 branch codes. Branch codes can be in any order and must be uppercase. The end of the BRANCHES array is indicated by a blank array element, which must be present. For example, BR/A2(3) 'A' 'CD' ' '. If ACTION = 2, this argument can be any An field: it must be present, but its contents are not used.

H.13.14 SETKEY - Simulate Keystrokes in TRANS

The SETKEY subroutine provides a way for the RMO running with TRANS to simulate a specified series of keystrokes being typed at the keyboard.

SETKEY can "pause" during the simulated series of keystrokes. This capability is useful to simulate "human speed" data entry, or simulate the user pausing to read from the display in the development of benchmark screens.

H.13.14.1 SETKEY Syntax

SETKEY's sole argument is the specification of the sequence of keystrokes to be simulated by TRANS. This sequence is usually specified as a character string contained in an alphanumeric field or constant. However if the TRANS environment file contains the line "setkey=physical" then the sequence can be specified as an integer array (see Section 6.17.6 “SETKEY=PHYSICAL, Simulate VT-type Function Keys”).

```
NULL = SETKEY(AR)
```

NULL/I	Required for syntax purposes only.
AR/An	An Alpha string that describes a series of keystrokes, where: A single character (separated by blanks) represents that character. A string of more than one character represents a standard function key name. A string that begins with "%" is a TRANS keystroke function. A string that begins with "-" followed by number will cause TRANS to pause that number of hundredths of a second.
AR/I(n)	(Used when "setkey=physical" is present in TRANS$ENV, to specify keystrokes by the physical "escape sequences" sent when they are pressed). Each element of the array is the integer decimal code of an ASCII character (see table in Appendix H.2). The integer array is read until a element that contains zero (0) is encountered.

If the statement "setkey=physical" is present in the TRANS environment file (see Section 6.17 “The TRANS Environment File”), TRANS checks the array to see if escape sequences sent by any VT function keys are present (one element of the array for each character of the escape sequence). If so TRANS will simulate the VT function keys corresponding to that escape sequence.

If an element of SETKEY's array is a negative integer (range 1 to 127), TRANS will pause that number of hundredths of a second at that point in the sequence of keystrokes.[24]

24. UNIX systems can pause only in full second increments of time. On UNIX systems the time paused is rounded up to the next highest full second.

H.13.14.2 SETKEY Example

For example:

```
LOCAL
AR/A60 '%exit D A T A CR 2 5 CR'
NULL/I
PROGRAM
NULL = SETKEY(AR)
```

When SETKEY is called TRANS will simulate the keystrokes "EXIT", "D", "A", "T", "A", "RETURN", "2", "5", and "RETURN" being typed the next time it reads the keyboard. EXIT will cause TRANS to clear the screen and prompt for a new screen name. "D A T A RETURN" will be the answer to the prompt, which will cause TRANS to load DATA.TRO and display the top-of-file record. "2 5 RETURN" will cause TRANS to display the record with key value 25 in the new screen loaded from DATA.TRO.

H.13.15 MOVFLD - Move Fields Among Files Accessed via TRO

The MOVFLD subroutine moves the contents of fields between external files in a TRS, and/or between an external file and the active file. This subroutine is useful for moving large numbers of fields from one file to another, eliminating lengthy LINK and/or APPEND paragraphs.

The data is moved from the "from" file to the "to" file. MOVFLD moves a series of fields to the "to" file, based on field name matches with fields in the "from" file. Some or all of the "from" file fields may be local field names from the screen definition (TRS). The fields in the "to" file must have the actual names of the "to" file definition. The "from" fields must have identical names and types to the "to" fields. The movement of field values from the "from" file record buffer to the "to" file record buffer is initiated by the RMO behind the screen.

MOVFLD **forces** the write-back at end of record processing to "to" LINK files even if no explicit LINK field is changed. Explicitly stated LINK fields ("L" fields in the LINK paragraph) may also be changed by regular LINK functionality. LINK key field(s) must be in a LINK Paragraph.

If an APPEND file is the "to" file, the APPEND condition letter must be set to write the record back to the APPEND file. The key field(s) in the APPEND file must be in the APPEND paragraph.

Use MOVFLD cautiously! Fields in a "to" file may be changed even though they are not mentioned in the TRS or RMS.

H.13.15.1 MOVFLD Syntax

```
STAT = MOVFLD(FROMFILE,TOFILE,FIRSTFLD,LASTFLD,DUMMY)
```

STAT/I	Status of the operation requested.
1	the operation was successful
-1	the “from” file was not found
-2	the “to” file was not found
-3	the first field was not found in the “to” file

STAT/I	Status of the operation requested.
-4	the last field was not found in the "to" file
-5	the first field does not precede the last field
-6	overflow of the internal move table; MOVFLD can manage about 300 fields
-7	the "to" file does not contain a particular field, or the fields do not match in type
-8	a text field (TI or TX) is included in the list of fields to be moved, MOVFLD does not support movement of text fields.
-11 -12...	If the absolute value of STAT is 11 or greater, then MOVFLD is unable to move a particular field. Instead MOVFLD returns diagnostic information on the error-causing field. The "problem" field in nth field being moved into the "to" file where n is the absolute value of STAT-10. For example, if STAT returns -17, then the field in question is the 7th field, or if the STAT returns -21, then the 11th field is causing the error condition.
FROMFILE/An	File the data is being moved from.
TOFILE/An	File the data is being moved to.
FIRSTFLD/An	The first field in the "to" file to be moved. FIRSTFLD must be an actual field in the "to" file.
LASTFLD/An	The last field in the "to" file to be moved. LASTFLD must be an actual field in the "to" file.
DUMMY/I	Required by syntax. Not used.

H.13.15.2 MOVFLD Example

Consider the following file definitions:

```
*     PO.MAS
MAS 1000
PO#       X999999 KEY1
FUND      X9
PROG      X99
OBJ       XA99
ITEM#     A20
QUANTITY  I
AMT       D2
...
*     LOG.MAS
MAS 100
PO#       X999999 KEY1
OLDFUND   X9
OLDPROG   X99
OLDOBJ    XA99
FUND      X9
PROG      X99
OBJ       XA99
ITEM#     A20
QUANTITY  I
AMT       D2
```

If the purchase order file changes such that the original FUND, PROGram, or OBJect billed for item is changed, then an APPEND paragraph may be used to log the changes. By using the MOVFLD subroutine in the RMO to move actual and local fields from the active file to the APPEND file, the user writes a simple APPEND paragraph to log all of the fields in LOG.MAS.

```
*   POCHG.TRS
P PO.MAS 1 POCHG.RMO
APPEND LOG.MAS ACTION A
PO#
END
...
*   POCHG.RMS
*
FILE PO.MAS
LOCAL
M$M/A2
S$S/A6
OLDFUND/X9
OLDPROG/X99
OLDOBJ/XA99
STAT/I
FROMFILE/A20 'PO.MAS'
TOFILE/A20   'LOG.MAS'
FIRSTFLD/A10 'OLDFUND'
LASTFLD/A10  'AMT'
DUMMY/I 0
ACTION/A1
PROGRAM
IF M$M NE 'UP' THEN STOP ; END
*
* record original values
*
IF S$S EQ 'BEGREC' THEN
   OLDFUND = FUND ; OLDPROG = PROG ; OLDOBJ = OBJ ; STOP ; END
*
* IF VALUES CHANGE, APPEND A NEW RECORD WITH ALL LOG.MAS FIELDS
*
IF (S$S EQ 'FUND' OR 'PROG' OR 'OBJ') AND
   ((FUND NE OLDFUND) OR (PROG NE OLDPROG) OR (OBJ NE OLDOBJ))
  THEN STAT = MOVFLD(FROMFILE,TOFILE,FIRSTFLD,LASTFLD,DUMMY)
   ACTION = 'A' END
```

If the value for the FUND, PROG or OBJ is changed by the user, MOVFLD moves the values from the PO.MAS fields and from the local fields into the APPEND file, LOG.MAS. The key field, PO#, is included in the APPEND Paragraph. The condition field, ACTION is set to A to perform the append to LOG.MAS after the RMO returns to TRANS.

H.13.16 FLDINFO - Retrieving Information About Fields in TRANS

The FLDINFO subroutine retrieves information about fields that are present in the virtual record on which a screen operates. The general syntax is:

```
STAT = FLDINFO(WHAT,SEQ,FLDNAM,TYPE,Y,X,FLAGS)
```

where:

WHAT/I	Mask to indicate which field types we want information about. The following flags are defined: 0: All field types 1: Key fields 2: Editable fields 4: Display fields 8: Virtual fields 256: Information about fields on a multi-record line only 1024: Return pixel values for Y and X in FLAGS(1) and FLAGS(2) -1: Get information about field in FLDNAM. These flags can be combined by adding them together.
SEQ/I	To start, set SEQ = 0. The call is returned with the relative sequence number of the field in the internal list of file. On the next call to get the next field that qualifies,leave it unchanged.
FLDNAM/An	Field to return the name of the field. Normally A18.
TYPE/An	Field to receive the base type of the field. If TYPE is A20 (or larger) the full data type will be returned (e.g. A32, D2, X9999), while if TYPE is shorter, e.g. A4, only the base data type is returned (i.e. 'A' for An fields, 'X' for picture fields etc.).
Y/I	Line number where field is displayed, or zero.
X/I	Column number where field is displayed, or zero.
FLAGS/I(n)	An integer array to receive further information about the field. FLAGS(1) 1 if field has lookup FLAGS (2) 1 if field is tied to combo box
STAT/I	Returned with 1 if a field is found, else 0 (end of list). If **WHAT = -1** (get information about a specific field) **STAT** is set to **1-8** to indicate what type of field it is (1 = Key, 2 = editable...). Observe that 3 means editable key field, 5 means display only key field, etc.)

The **FLDINFO** subroutine can also be used to convert a line/column position in the current window to screen pixel coordinates. By setting **WHAT = 1024** and **Y = line number** and **X = column number**, screen coordinates are returned in **FLAGS(1)** (Y coordinates) and **FLAGS(2)** (X coordinates).

H.13.17 APVDLG: Activate an APV Dialog Box From the RMO

The APVDLG subroutine activates the APV dialog box feature from the RMO, allowing for more complex dialog boxes than the DLGBOX subroutine. The syntax is:

```
STAT = APVDLG(MSG,TARGETS,APVPATH,INSTANCES[,OPTIONS])
```

MSG/An [(dim)]	The text to appear on top of the dialog box. Up to four (4) lines may be entered (e.g. MSG/A40(40)).
TARGETS/An(d)	An alphanumeric array containing the names of the fields that are to receive values form the dialog box. The data type of the fields must correspond to the FORMAT statement in the APV file.
APVPATH/An (d)	An alphanumeric array containing the pathnames of the APV files to use for each target field (e.g. MYAPV: fund.apv).
INSTANCES/I(d)	The instance number to use in each APV file. If an instance is missing, or zero, it will be set to 1.
OPTIONS	Optional argument specifying optional behavior: 1 Return values in the TARGETS fields. This is the default if the OPTIONS field is not present. 2 Create the logical names from the APV files, and do not return any values (the TARGETS field may be a dummy). 3 Both. Create the logical names and return the values in the TARGETS fields.
STAT/I	1 = OK 0 = Cancel pressed. -1 = Too few arguments -2 = A maximum of 20 target fields are allowed. -3 = Too few APV pathnames -4 = Too few pathnames -(100 + N) = Target field n not found -(200 + N) = Missing APV file for target n -(300 + N) = Error in APV file for target n -(400 + N) = Data type mismatch between target n and APV

H.13.18 MENUBAR - AppMenu Manipulations

AThe MENUBAR subroutine allows you to disable (gray) and enable (un-gray) menu bar items implemented in the AppMenu sub-system and to execute menu item tasks from the RMO. The syntax is:

```
STAT = MENUBAR(OPER,FLAG,ITEMS,[TARGETS])
```

OPER/I	Operation to perform: 1 = Enable items 2 = Gray (disable) items 4 = Execute an AppMenu item 8 = View popup menu for the item

FLAG/I	Mask to modify execution 0 = Default 1= For OPER =4, check if user can run task 2 = For OPER =4, work the APV dialog box, but do not execute the command. 4 = For OPER =4, execute the menu item using saved values without re-prompting the user.
ITEMS/I(n)	A list of UNIT type identifiers of menu items to put in state OPER. A zero ID will terminate the list (a maximum of n items will be processed).
TARGETS/An(n)	Only valid for OPER = 4 and FLAG = 2. An array containing the names of the fields to receive the values from the dialog box.
STAT/I	Return status. 1 = OK -1 = Invalid number of arguments -2 = Invalid OPER value -3 = Invalid FLAG type **Returned by OPER = 4: Execute:** 1 = OK, task type was MENU 2 = OK, task type was SCR 3 = OK, task type was REP 4 = OK, task type was COM 5 = OK, task type was HLP 6 = OK, task type was WHLP 7 = OK, task type was PERL 8 = OK, task type was EXEC -4 = No such menu item or task found -5 = Task is disabled -6 = Task has error flag set -7 = Task is in use -8 = User does not have access to task -9 = User name not of type An -10 = Error parsing parameter file for menu item -11 = Target array not of type An -(100 +n) = Target field **n** not found -(300 +n) = Error in APV file for target **n** -(400 +n) = Data type mismatch for target **n** **Returned by OPER = 4, FLAG = 1 (Check if user can execute):** 1 = OK, task can be executed -4 = No such menu item or task can be found -5 = Task is disabled -6 = Task has error flag set -7 = Task is in use -8 = User does not have privilege to execute task

The return status of (-4) is only returned if no menu items of the requested type exists (e.g. UNIT = 0 and no AppMenu menus are defined). If e.g. a list of task ids are given, and some of those task ids are not found in the current AppMenu menus, these task

IDs are silently discarded. This allows the RMO to gray a number of menu items that for some reason should be disabled without having to worry about if the current user actually have these items in his/her menu.

H.13.18.1 Using MENUBAR to enable/disable Toolbar Buttons

The MENUBAR subroutine can enable or disable toolbar buttons. The syntax is:

```
STAT = MENUBAR(WHAT,8,BTNS)
```

where:

WHAT	1=Enable 2=Disable
OPER	8=Modify Toolbar
BTNS/I(n)	An integer array containing ids of toolbar buttons to enable or disable. The button ids are those described in Section 6.17.13.10 "Toolbar", e.g. 12 = Lookup, 14 = Print, 27 = Calendar.

H.13.18.2 View Menubars

The MENUBAR subroutine may also be used to view the menu items specified in AppMenu files. It requires the logical name APPMENU_MENUSPEC, which identifies the AppMenu Specification file, and the logical name APPMENU_TASKFILE, which identifies the AppMenu Task file. See Appendix M: "The AppMenu SubSystem" for details. The syntax is:

```
STAT = MENUBAR(8,LEVEL1,LEVEL2,USERNAME[,Y[,X]])
```

LEVEL1/I	Level 1 ID of menu structure to view.
LEVEL 2/I	Normally zero, or level 2 ID if you want to view the menu structure from the second level.
USERNAME/An	The user or group name of the menu structure to view.
Y/I	Optional argument specifying the line number where the menu pops up.
X/I	Optional argument specifying the column number where the menu pops up.

STAT/I		Return status
	1	OK
	-9	USERNAME not alpha
	-10	Y is not of type integer
	-11	X is not of type integer
	-801	Logical name APPMENU_MENUSPEC not assigned
	-802	Logical name APPMENU_TASKFILE not assigned
	-803	Unable to open APPMENU_MENUSPEC file.
	-804	Field TSKID not found in APPMENU_MENUSPEC
	-805	The major key in APPMENU_MENUSPEC not alpha.
	-806	The minor keys in APPMENU_MENUSPEC are not integers.
	-807	The key requested in the MENUBAR call was not found in the APPMENU_MENUSPEC file.
	-808	Unable to open the APPMENU_TASKFILE file.
	-809	The TSKDESC field was not found in the task file
	-810	The TSKTYP field was not found in the task file.

H.14 Miscellaneous Subroutines

This "catchall" category of subroutines includes many different capabilities applicable in a variety of situations and useful across all ADMINS commands.

H.14.1 ASKSCR: Prompt directly from RMO

The ASKSCR subroutine displays a prompt or message on the screen. Optionally, ASKSCR accepts a response from the user, with or without echoing the response, without leaving the RMO. The response can be automatically converted to all uppercase.

The subroutine has a lot of uses, e.g., displaying various messages on the screen, prompting for passwords, verification routines, etc.

The syntax is:

```
STAT = ASKSCR(Y,X,PROMPT[,ANSWER][,OPTIONS],[COLS]]])
```

where[25]

STAT/I	-1: Invalid number of arguments -2: PROMPT and/or ANSWER not ALPHA fields -3: Invalid Y value -4: Invalid X value -5: ANSWER must be a field -6: OPTIONS and COLS must be integers -7: Box option invalid. TRANS only & COLS required. `>=0: Length, in bytes, of ANSWER` -100: Cancel button pressed
Y/I	(field or constant) Line number to prompt at.
X/I	(field or constant) Column number to prompt at.

PROMPT/An	(field, subscripted field, or constant) Prompt string If one of the prompt fields starts with '%%', the rest of this field is used as caption text in the dialog box, e.g: PROMPT/A60 (3) 'This is prompt line' 'And this is prompt line 2' '%% Caption Text' ASKSCR retrieves messages from the ADM$MESSAGEFILE file. If PROMPT arguments are passed in the form: #nn#, i.e. the message number preceded and followed by the "#" character, ASKSCR will get the message for that number and use it for that prompt.
ANSWER/An	(must be a field) If the 4th parameter given is an alpha field, ASKSCR waits for a response which is loaded into this field. If the 4th argument given is not an alpha field, ASKSCR does not wait for a response, and the argument is treated as OPTIONS (described below).
OPTIONS/I	(field or constant) Any desired combination (sum) of these flag values: 1 Do not echo ANSWER 8 Convert ANSWER to uppercase.

25. When the PROMPT argument references an alpha array using a constant subscript, the constant must be explicitly typed: e.g., PROMPT('2/I') rather than PROMPT(2).

H.14.2 CHECKFILE - Check Whether File Exists

Use CHECKFILE to determine whether or not a file exists. The syntax is:

```
STAT = CHECKFILE(FILENAME)
```

STAT/I	0: File not found 1: File exists
FILENAME/An	(Field, array, or constant) File specification (may use full array to specify file). If this argument begins with two question marks, e.g.: `??MY_LISFILES:*_JOHN.LIS` then CHECKFILE will search using the wildcarded string that follows the "??", returning 1 into STAT if any files match and 0 otherwise. If this wildcard syntax is used, and this argument is an array rather than a simple field, CHECKFILE will load the array (until all elements are used) with all the wildcard matches found, and STAT will be set to the number of array elements that have been loaded. Note that the number of possible matches may exceed the number of array elements reserved to hold them.

CHECKFILE supports File Open Dialog Boxes (see Appendix H.12.1.1 "File Open Dialog Box" for more information).

H.14.3 SAVEAS -Save File Dialog Box (with suggested name)

The SAVEAS subroutine is used to launch a dialog box which prompts to save a file to a certain directory under a suggested or a user supplied name. The syntax is:

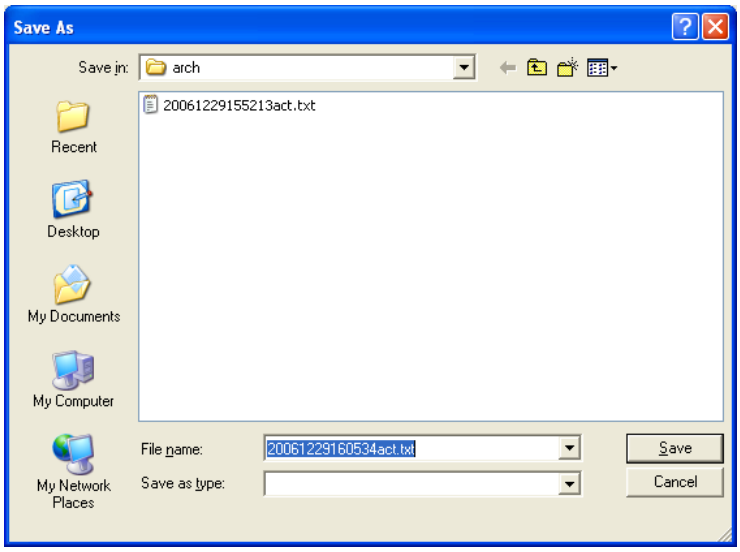
```
STAT = SAVEAS(FILNAM,DIRNAM[,OPTION[,SUGNAM ]])
```

where

FILNAM/An(N)	(Alphanumeric field or array) The pathname of the file to be saved in another location. It may be or contain a logical name.
DIRNAM/An	The pathname of the directory where to store the SaveAs file name.
OPTION/I	(Optional) Options currently implemented: 1 = User cannot change file type
SUGNAM/An	(Optional) Field containing suggested name for file to be saved.

STAT/I	Return status. Possible values are 1 = OK, file saved -1 = Too few arguments -2 = File does not exist, or not accesable -7 = Copy failed 9 = User cancelled the operation

H.14.3.1 SAVEAS example




```
file activity_file
stat/i
m$m/a2
s$s/a16
today/dt
now/a8
out/a2(3)
alias out
path/a40 'activity.txt'
dir/a40 'arch_dir'
fnm/a40
alphadt/a20
fmtdat/a10 '..Y4MD'
sep/a1 ':'
program
*
* if launch field is entered save a copy of the activity file
*
if m$m ne 'UP' then stop ; end
if s$s ne 'LAUNCH' then stop ; end
*
* get a string version of the date in Y4MD form
*
alphadt = fcat(alphadt,fmtdat,today)
*
* remove the colons from the current time stamp
* (split the string on colons into an array)
*
stat = split(now,sep,out)
*
* build a suggested file name with the date and time
*
stat = format('****act.txt',alphadt,out_1,out_2,out_3,fnm)
*
* launch the saveas dialog box
*
```

```
stat = saveas(path,dir,1,fnm)
```

H.14.4 CLIPBOARD - Accessing the Windows Clipboard

The CLIPBOARD subroutine both reads and writes data in the clipboard. The syntax is:

```
STAT = CLIPBOARD(OPER,FLAG,BUF)
```

where:

STAT/I	1 OK 2 OK, data lost (buffer to small, or record too small). 0 No data available -1 Unknown operator (not G, P, Q or E) -2 Unable to open the clipboard -3 BUF has invalid type -4 Previous operator was not G (GN only) -5 Internal error (unable to allocate memory)
OPER/A2	'G' - to Get data from the clipboard (read) 'GN' - to Get the next chunk of data if the first Get could not retrieve all the data (STAT = 2 on return). 'S' - to Store data which is written to the clipboard by a 'P' call. Each subsequent 'S' call (and the terminating 'P' call) appends its content to what was previously stored. No data is written to the clipboard if the 'S' calls are not followed by a 'P' call. 'P' - to Put data on the clipboard (write). All from any previous 'S' calls, and the data included in the 'P' call, will be put on the clipboard. A Put always empties the clipboard before writing to it. 'Q' - to Query if there is text data available. 'E' - to Empty the clipboard. Only necessary if you want to remove what was previously put on the clipboard.

FLAG/I	Flag to modify the behavior of CLIPBOARD. The FLAG is a mask, and may be or-ed together. Returned with the total number of bytes transferred. **Put:** 1: Output each element of BUF as a separate record (separated by CR/LF). 2: Treat two consecutive BUF elements as one record. 4: Treat four consecutive BUF elements as one record. 8: Treat eight consecutive BUF elements as one record **Get:** 1: Treat each line in the clipboard separated by a CR/LF pair as a separate record, and start on a new BUF boundary after each CR/LF. Any data that does not fit in a BUF record is lost. 2: Treat two consecutive BUF elements as one record. 4: Treat four consecutive BUF elements as one record. 8: Treat eight consecutive BUF elements as one record FLAG is set to the number of characters returned on stream input, and to the number of records returned on record input (FLAG = 1 set when called).
BUF/An(dim)	Buffer with data to put on the clipboard, or to receive data from the clipboard. BUF may be used as one contiguous stream of data, or each element (DIM(n)) may be considered a separate record (line) of data on the clipboard (on the clipboard separated by CR LF). On Get (input) the default is to stream the data and throw away any CR/LF encountered. FLAG = 1 modifies this, inputting one record into each BUF element, where any excess data in each record is lost. On Put (output) the default is to stream the output with no CR/LF inserted. FLAG = 1 modifies this and treat each element of BUF as a separate record, inserting a CR/LF after each element.

H.14.5 DCS: Date to Year/Week/Day; Check Digit Conversion

The DCS subroutine performs several different functions:

1. Converts an ADMINS Date to Year, Week and Day, etc.
2. Determines if a given day has been defined as a holiday
3. Determines the occurrence of a weekday within the month for a date

4. Finds the date of the specified occurrence of the specified weekday in a specified month/year.
5. Computes a date a specified number of business days after a specified date
6. Computes a Base 10 Check Digit
7. Verifies a Base 10 Check Digit
8. Tests Check Digit for Norwegian Social Security Number

H.14.5.1 DCS Syntax: Convert Date to Year, Week, Day

```
STAT = DCS(OP,DATE,RES,OCCUR)
```

OP/I		Operation
	1	Convert DA field
	2	Convert DT field
	3	Convert DA with holiday status
	4	Convert DT with holiday status
	101	For DA field. Also load OCCUR field with occurrence of day-of week within month
	102	For DT field. Load OCCUR as above
DATE/DA or DT		Contains the ADMINS date to be converted.
RES/I(7), or RES/I(8) for OP=3, 4		Will contain the result from the conversion: RES(1) = Year for week number RES(2) = Week number RES(3) = Day of week, where 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday,6=Saturday, 7=Sunday RES(4) = Year (of date) RES(5) = Day number within year RES(6) = Month RES(7) = Day of month RES(8)= Holiday? (1=Yes, 0=No) for OP=3, 4
OCCUR/I		Occurrence of day-of-week within month (set by OP=101,102)
STAT/I		will be returned as: 1 = OK -1 = Invalid ADMINS date in the DA or DT field -99 = Unknown operation

The routine has two year fields, because the first days of a year may belong to the last week of the previous year.

H.14.5.1.1 Date Conversion Example

Given an ADMINS Date, find the number of the day in the year and the name of the day:

```
* Convert DATE to Day of Year and Name of Day
FILE DATA.MAS
LOCAL
NULL/I
STATUS/I
RESULT/I(7)
ADAY/A10(7) 'Monday' 'Tuesday' 'Wednesday' 'Thursday'
  'Friday' 'Saturday' 'Sunday'
PROGRAM
*
* First argument is 1 for conversion of DA field
```

```
STATUS = DCS(1,DATE,RESULT,NULL)
*
* If invalid get out
IF STATUS EQ -1 THEN GOTO DONE END
*
* Get DAYNUMber and determine the NAME  of the Day
DAYNUM = RESULT(5) ; NAME = ADAY(RESULT(3))
*
DONE: STOP
*
```

H.14.5.2 DCS Syntax: Get Date of Nth weekday in a month

To get the Nth occurrence of a weekday in a specified month, the syntax is:

```
STAT = DCS(OP,DATE,AR,NULL)
```

where OP = 201 for DA fields, and 202 for DT fields. The date of the Nth occurrence is returned in DATE.

OP/i		Operation
	201	put occurrence in DA field
	202	put occurrence in DT field
DATE/DA or DATE/DT		Returns the ADMINS date requested.
AR/I(4)		AR(1): YEAR AR(2): MONTH AR(3): Day of week, where 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday,6=Saturday, 7=Sunday AR(4): Requested Occurrence
NULL/I		

For example, to find the date of Thanksgiving Day in the U.S.A. (the fourth Thursday in November) in 2008, use the following:

```
AR(1) = 2008 ; AR(2) = 11 ; AR(3) = 4 ; AR(4) = 4
STAT = DCS(201,DATE,AR,NULL)
```

DATE will be loaded with the value “27-NOV-08”.

H.14.5.3 DCS Syntax: Compute date some number of business days after another date.

To get the date that falls a given number of business days[26] after a specified date, use this DCS subroutine syntax:

```
DCS(OP,STARTDATE,BIZDAYS,ENDDATE)
```

26. Business days are Monday through Friday, unless the day is specified to be a holiday. See Appendix 5.5.19 “Calendar Keyword” for an explanation of how to define holidays. If the start day designated is not a business day, the date is calculated by first moving to the first business day after the specified start date, and then moving forward the specified number of business days.

where: .

OP/i		Operation
	301	calculate for DA fields
	302	calculate for DT field
STARTDATE/DA or DT		Start date for calculation
BIZDAYS/I		Number of Business days to be added
ENDDATE/DA or DT		(Returned)

For example, to find the date 2 business days after "August 11, 2007" use the following:

```
DTDATE/DT '11-AUG-2007'
DTBIZDAYS/I 2
THISDTDATE/DT

STAT = DCS(302,DTDATE,DTBIZDAYS,THISDTDATE)
```

THISDTDATE will be loaded with the value "15-AUG-2007".

H.14.5.4 DCS Syntax to Compute Base 10 Check Digit

```
STAT = DCS(9,INP,RES,LEN)
```

INP/X9-9	Contains the value for which the DCS should compute the Check Digit.
RES/X9	Will contain the resulting Check Digit
LEN/I	Number of digits in INP furnished by user.
STAT/I	Will be returned as: 1 = OK -1 = Error in INP format -99 = Unknown operation

H.14.5.4.1 Compute Check Digit Example

Given a number compute the check digit and then combine the check digit with the number:

```
* Compute check digit on NUMBER/X999999999
* Create new number CHECKNO with check digit at end
FILE DATA.MAS
LOCAL
LENGTH/I 9
STATUS/I
RESULT/X9
CHECKNO/X9999999999
PROGRAM
* Get check digit, first argument is 9
STATUS = DCS(9,NUMBER,RESULT,LENGTH)
*
* If error, get out else create full number with check digit
IF STATUS LT 0 THEN GOTO DONE ELSE
   CHECKNO = CCAT(CHECKNO,NUMBER,RESULT) END
*
DONE: STOP
```

H.14.5.5 DCS Syntax to Verify Base 10 Check Digit

```
STAT = DCS(10,INP,NULL,LEN)
```

INP/X9-9	Contains the value for which the DCS should compute and verify the Check Digit. Last digit is Check Digit.
NULL/x	Dummy argument for syntax reasons only.
LEN/I	Number of digits in INP furnished by user.
STAT/I	Will be returned as: 1 = OK -1 = Check Digit incorrect -99 = Unknown operation

H.14.5.5.1 Verify Check Digit Example

```
* Verify check digit of CHECKNO/X9999999999 and extract NUMBER
FILE DATA.MAS
LOCAL
NULL/I
LENGTH/I 10
STATUS/I
PROGRAM
*
* Verify check digit, first argument is 10
STATUS = DCS(10,CHECKNO,NULL,LENGTH)
*
* If error, get out else extract NUMBER
IF STATUS LT 0 THEN GOTO DONE ELSE
   NUMBER = STR(NUMBER,CHECKNO,'1/I','9/I') END
*
DONE: STOP
```

H.14.5.6 DCS Syntax: Test Check Digit, Norwegian SS#

```
STAT = DCS(11,FNR,NULL,NULL)
```

FNR/X99999999999contains a Norwegian SS#

STAT/I	Will be returned as: 1 = FNR valid -1 = FNR invalid -99 = Unknown operation

H.14.6 DLGBOX - Prompt for Several Values in One Dialog Box

The DLGBOX subroutine displays and prompts for several different values (up to 20) in a single dialog box.

DLGBOX supports "automatic" lookup for fields that are tied to a codelist[27].

DLGBOX can also retrieve messages from the ADM$MESSAGEFILE file for the CAPTION and PROMPTS. If these arguments contain a string are in the form: ***#nn#*** DLGBOX will try to retreive message number ***nn*** and use it for the corresponding caption or prompt.

The syntax is:

```
STAT = DLGBOX (CAPTION,PROMPTS,FIELDS[,OPT])
```

where:

CAPTION/An	What appears in the banner
PROMPTS/An(d)	An alphanumeric array containing the prompts, or label text, to appear in front of each field.
FIELDS/An(d)	An alphanumeric array containing the name of the fields to be loaded with the values that are prompted for. DLGBOX keeps adding fields until the full array is used, or an empty (blank) array element is found. Values entered in fields are checked for proper format.
OPT/I(d)	An optional integer array containing field options. Available options are: 1: Do not echo the characters typed (* are echoed) 2: Uppercase the input 4: Lowercase the input 8: Do not check against codelist table 16: Error if present in codelist table 32: Display codelist description when codelist value entered (if field tied to codelist with defined lookup).
STAT/I	1 = OK 0 = Cancel pressed. -1 = Too few arguments -2 = FIELDS must be an array -3 = Too many fields (max 20) -10 = Unable to create dialog box -10n = Field n in FIELDS array not found

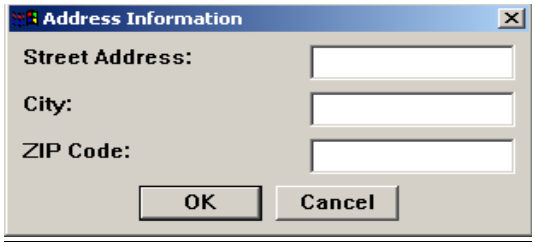
The following example demonstrates this subroutine in use:

```
CAPT/A20 'Address Information'
LBLS/A20(4) 'Street Address' 'City:' 'ZIP Code:'
FLDS/A18(4) 'STREET'         'CITY'  'ZIP' ' '
…
STAT = DLGBOX(CAPT,LBLS,FLDS)
```

The following dialog box displays[28]:

27.If any field has an automatic lookup specified in the data dictionary then a lookup button will be displayed in the dialog box. When a field with an automatic lookup is in focus the Lookup window will activate when the button is clicked (if the field does not have an automatic lookup the Lookup button is disabled (grayed). .

28.The OK and Cancel buttons, and the Lookup button (if in use) can be activated via Windows accelerator keys



H.14.7 EVALUATE - Compile, Execute Expression at Runtime

The EVALUATE subroutine is used to compile and/or execute, at run time, an ADMINS expression stored in an alphanumeric data field or local array.

First, the expression (e.g. 'BAL GT 0' or 'A = B + C') is placed in an alphanumeric (An) field or local array, and EVALUATE is called to store the expression. Later on, EVALUATE is called to compile and/or execute the stored expression.

The expression given to EVALUATE can use any existing fields and can be arbitrarily complex. It must be either a Boolean statement (one which could be used with SELECT in REPORT, for example), or else an assignment statement of the form "FIELD = expression" (the '=' in assignments must appear in the first line of the expression).

EVALUATE executes a single, self-contained expression, not multiple expressions: the pair of expressions 'A = 1 ; B = 2' can't be executed by EVALUATE. IF statements cannot be used because they involve at least two expressions (IF expression THEN expression END).

H.14.7.1 Using EVALUATE

Using EVALUATE is a two-step process: an initial EVALUATE call stores the expression, subsequent calls compile and/or execute the expression.

First, store the expression:

Place the text of the expression in an alphanumeric (An) field or local array. An expression can have more than one line if it is placed in an array and the ':' continuation syntax is used. For example, the expression 'X GT 0 AND X NE Y' could be supplied to EVALUATE as two lines in an array:[29]

```
EXP(1) = 'X GT 0 :'
       and
EXP(2) = 'AND X NE Y'
```

Call EVALUATE with FUNCTION = 1 to store the expression text:

```
STAT = EVALUATE(1,EXP).
```

Second, execute the stored expression (FUNCTION = 2).

```
STAT = EVALUATE(2).
```

The first time EVALUATE is called with FUNCTION = 2, the stored expression is compiled;[30] and, if it compiles successfully, it is executed. Subsequent calls with FUNCTION = 2, do not compile the expression again: it is just executed.

---

29. If an array is used for the expression text, it must be a local array in an RMO.

Alternatively, just compile the stored expression (FUNCTION = 3).

```
STAT = EVALUATE(3).
```

This makes it possible to perform syntax checking without actually executing the expression, which might change data. If the expression compiles and you then use FUNCTION 2, the expression is not compiled again, it is just executed.

Whenever you call EVALUATE with FUNCTION = 1 to store a new expression, the next EVALUATE FUNCTION = 2 call compiles the stored expression before executing it. In TRANS, the stored expression is compiled at the first EVALUATE(2) call after each branch. Thus, in TRANS, you can have one screen where an expression is entered and stored, and can then branch and execute the expression in one or more other screens.

Syntax checking of the expression is not done until the expression is compiled.[31] It is important to check the status return of EVALUATE(2) and EVALUATE(3) calls and take appropriate action if there is an error status (e.g., automatically branch back to the expression entry screen).

The expression text can be entered in lowercase or mixed case characters. EVALUATE internally converts everything in the expression to uppercase except constants which are surrounded by apostrophes.[32]

H.14.7.2 EVALUATE Syntax

```
STAT = EVALUATE(FUNCTION[,EXPRESSION])
```

FUNCTION/I	Function code (field or constant): 1: Store expression text 2: Compile and/or execute stored expression 3: Compile stored expression (do not execute).
EXPRESSION/An	If FUNCTION is 1, alpha field or local alpha array containing text of expression to store. EXPRESSION text need not be uppercase. If FUNCTION is 2 or 3, this argument need not be present (if present, it is ignored).

30. Compiling an expression at run time is not in general an efficient method of doing things; so don't use EVALUATE unless you really need it! Once the expression is compiled, however, any subsequent FUNCTION = 2 calls of EVALUATE execute almost as efficiently as if the expression had been coded in an RMS.
31. The expression given to EVALUATE should not call EVALUATE. Other ADMINS subroutines can be used freely in the expressions given to EVALUATE.
32. This case insensitivity is intended to make it easier to enter valid expressions; but users should always enclose alphanumeric or picture constants in apostrophes: if not, these constants will be converted to uppercase, which may not be desired.

STAT/I	Normal status values:
	1: If FUNCTION is 1, successful load.
	If FUNCTION is 2: Boolean expression is TRUE; or non-Boolean expression was evaluated.
	If FUNCTION is 3, expression compiled.
	0: FUNCTION is 2 and Boolean expression is FALSE
	Error status values:
	-101: FUNCTION is not 1, 2 or 3
	-102: FUNCTION is 1 and there are not two arguments
	-103: EXPRESSION contains a blank line
	`-104: ':' continuation at end of EXPRESSION field`
	`-105: FUNCTION is 2 and no expression was stored`
	`-106: EVALUATE attempted to call itself`
	`-107: EXPRESSION exceeds EVALUATE limits`
	other values LT 0: EXPRESSION line# (negated) with syntax error

A call to EVALUATE with FUNCTION = 2 requires 300 words in the DA array. EVALUATE never requires more than 300 DA words, no matter how often it is called. At a branch in TRANS, any DA space used by EVALUATE is freed up.

H.14.8 EXTERNAL - Call External Language Routine

The EXTERNAL subroutine supports the ADMINS External Language Facility,[33] which makes it possible to call routines written in C, FORTRAN, MACRO, etc., directly from an ADMINS command.

H.14.8.1 EXTERNAL Syntax

The syntax for the EXTERNAL subroutine is:

The EXTERNAL subroutine can be called with up to 16 arguments (the arguments must be field names, not constants). The EXTERNAL arguments can have any data type.[34]

33. The ADMINS External Language Facility is designed for use by experienced 3GL programmers in situations which require specialized data handling. Nevertheless, the ELF is a relatively straightforward interface which imposes few if any limitations on the nature of user written routines. The external, user-written routines must be linked as a shareable image with ADMINS ELF module EXTERN.OBJ. Complete documentation for the ADMINS External Language Facility is on the distribution tape.
34. Arrays are handled in a special way, see Appendix H.14.8.2 "Passing Arrays as Arguments"

```
STAT = EXTERNAL(FIELD_1[,FIELD_2,...,FIELD_16])
```

STAT/I	Return status code 1 Success 2 Warning: 7-bit nonprinting character in data 4 Warning: 8-bit nonprinting character in data 6 Warning: 7-bit and 8-bit nonprinting characters -1 Error: called with no arguments -2 Error: unknown data type -3 Error converting to ASCII format -4 Error: null terminator was overwritten -5 Error converting from ASCII format -6 Cannot create dump file -7 Cannot allocate memory for buffer -8 Error: number of elements in USER$ARRAY is out of range. -9 Error: USER$ARRAY not Integer field type
FIELD_1...	Up to 16 fields with any data types. Constants cannot be used as arguments. EXTERNAL requires at least one argument.

When EXTERNAL is called, ADMINS converts the EXTERNAL subroutine arguments to an ASCII representation in a buffer, and passes the buffer[35] to the user-written external routine.

The data buffer contains an ASCII representation of each EXTERNAL argument, in the same order in which the arguments appear in the EXTERNAL call in the RMS. The ASCII representation of each argument in the buffer has a fixed length, determined by the ADMINS field type of its corresponding argument, and is also null (zero) terminated. There is an additional zero after the last argument's representation in the buffer.

When the external routine returns, ADMINS examines the buffer. If any values have been changed by the user written routine, they are converted back to the appropriate ADMINS field types and the field referenced by those arguments are updated.

The length and characteristics of the ASCII representation of each argument field-type in the EXTERNAL buffer is summarized in the following table.

Argument Type	Length (before null terminator)
I	6 characters, blank padded on right, no comma.
L	12 characters, blank padded on right, no commas.
Dn	17 characters, blank padded on right, no commas.
Fn	21 characters, blank padded on right, no commas.
DA	9 characters or maximum length for ADM$DATE format currently assigned, blank padded on right.
DT	11 characters or maximum length for ADM$DATE format currently assigned, blank padded on right
TM	11 characters.

35. Actually EXTERNAL passes the address of the buffer to the external routine.

Argument Type	Length (before null terminator)
An	n characters. N is always an even number. Blank padded on right.
Xpic	Size of picture.

For example:

```
APPLIC.RMS
----------
ROUTINE/I 10
AFLD/A8   'ABC'
DFLD/D2   123.45
IFLD/I    -10000

STAT = EXTERNAL(ROUTINE,AFLD,DFLD,IFLD)
```

```
The EXTERNAL data buffer could be represented as follows:

   Position             1         2         3         4
  in buffer:   01234567890123456789012345678901234567890
               ----------------------------------------
       Data:   10    |ABC     |123.45           |-10000||

              ('|' represents binary zero)
```

EXTERNAL has a debugging facility which dumps the contents of its buffer to a file. If one of the arguments to EXTERNAL is an An field which contains the string '?LIST?', ADMINS dumps the buffer before and after calling the external routine. The dump file is called EXTERNAL.LIS and is placed in the user's default directory. Since this option creates a new version of EXTERNAL.LIS at each call, it should be used sparingly.

H.14.8.2 Passing Arrays as Arguments

Array names can be used as arguments to EXTERNAL, but EXTERNAL must be told how many elements to process in each array. The special array USER$ARRAY/I(n) passes array size information to EXTERNAL.

If one or more arrays are to be passed as arguments, the arguments that are arrays must be consecutive in the EXTERNAL argument list, and they must be preceded by the USER$ARRAY argument, which describes them. For example:

```
APPLIC.RMS
----------
STAT/I
ROUTINE/I
IFLD/I
USER$ARRAY/I(3) 2 4 5
DARRAY/D(4)
XARRAY/X999(5)

STAT = EXTERNAL(ROUTINE,IFLD,USER$ARRAY,DARRAY,XARRAY)
```

USER$ARRAY must be an array of field type integer. USER$ARRAY(1) always contains the number of arrays which follow (in the example above, 2 arrays follow). USER$ARRAY(2) gives the number of elements in the first array; USER$ARRAY(3) gives the number of elements in the second array; etc.

USER$ARRAY itself is not placed in the data buffer which is passed to the user-written routine. The user written routine must either know the number of elements in array arguments, or else array sizes must be passed in separate arguments.

When arrays are passed, the EXTERNAL data buffer may become quite large. There is no fixed limit on its size: memory for the data buffer is dynamically allocated.

H.14.8.3 AdmExternal.c: Program Sample

The RMO subroutine EXTERNAL is implemented in ADMINS Win32 as a DLL (AdmExternal.dll), which must be present in the ADM_DIST directory. ADMINS, Inc. provides a model AdmExternal.dll that displays all the arguments to EXTERNAL in message boxes as an example of how to write your own EXTERNAL implementation (different implementation procedures may need to be used when installing this subroutine on your system).

The following is a sample AdmExternal.c program:

```
#include <windows.h>
BOOL APIENTRY DllMain(HANDLE hModule,
                      DWORD dwReasonForCall,
                      LPVOID lpReserved)
{
    switch(dwReasonForCall)
       {
       case DLL_PROCESS_ATTACH:
                break;
       case DLL_THREAD_ATTACH:
                break;
       case DLL_THREAD_DETACH:
                break;
       case DLL_PROCESS_DETACH:
                break;
       }
    return TRUE;
}
void adm_external(unsigned char *pubBuf)
{
   int              i;
   char             szCapt[40];
   unsigned char   *puc;

   for (i = 0, puc = pubBuf; *puc != '\0'; puc++, i++)
      {
      wsprintf(szCapt, "AdmExternal argumet %d", i + 1);
      MessageBox(NULL, puc, szCapt, MB_OK);
      puc += strlen(puc);
}
   return;
}
```

And the accompanying AdmExternal.def:

```
LIBRARY         AdmExternal
HEAPSIZE        1024
EXPORTS
```

```
adm_external    @1
```

You only have to modify the AdmExternal.c program to implement your own version of the EXTERNAL RMO subroutine, compile it, link it and replace the AdmExternal.dll in the ADM_DIST directory.

H.14.9 MSGBOX - Display Messages Requiring Only Acknowledgements or Predefined Answers

The MSGBOX subroutine displays messages that require only an acknowledgement or a predefined answer (e.g. OK, Yes, No).

The syntax is:

```
STAT = MSGBOX(TEXT[,CAPTION[,ICON[,BTNS[,DEFBTN]]]])
```

Where:

TEXT/An		What appears in the box If TEXT/An(dim) is an array, each element becomes a separate line to display. If TEXT is an array, an array element containing just '&&' will terminate the text displayed.
CAPTION/An		What appears in the banner.
ICON/I		Icon Type:
	1	Information (Default)
	2	Question
	3	Exclamation
	4	Stop
BTNS/I		Button Array Type:
	1	OK (Default)
	2	OK Cancel
	3	Yes No
	4	Yes No Cancel
	5	Retry Cancel
	6	Abort Retry Cancel
DEFBTN/I		Default Button:
		1, 2, or 3 (1 is default)
STAT/I		Return Value
	1	OK
	2	Cancel
	3	Abort
	4	Retry
	5	Ignore
	6	Yes
	7	No
	0	Unknown button
	-1	Argument 1 (TEXT) is wrong type
	-2	Argument 2 (CAPTION) is wrong type
	-3	Argument 3 (ICON) invalid value
	-4	Argument 4 (BTNS) invalid value
	-5	Argument 5 (DEFBTN) invalid value

(If ICON, BTNS or DEFBTN appears with a value of 0, the default value is used).

Both TEXT and CAPTION may be given as '#nnn#', where nnn is a message number from the ADM$MESSAGEFILE facility, see Appendix H.6.6 “GETMSG - Retrieving “Literal” Text From a File”.

H.14.10 REFRESH - Repainting an Area of the Window

The REFRESH subroutine forces an area of the window to repaint during a time when the screen does not normally refresh after being changed. The syntax is:

```
STAT = REFRESH(LINE,COLUMN,NLINES,NCOLUMNS)
```

where:

LINE/I	Line number at which to start. If LINE is zero, the whole window is repainted.
COLUMN/I	Column number at which to start.
NLINES/I	Number of lines to refresh. If NLINES is zero, the area from LINE all the way down to the bottom of the window is refreshed.
NCOLUMNS/I	Number of columns to refresh. If NCOLUMNS is zero, the area from COLUMN to the right of the window is refreshed.
STAT/I	Always 1.

H.14.11 GETJPI - Get Process Information

A GETJPI subroutine provides access to information about the current process.

The GETJPI syntax is:

```
STAT = GETJPI(ITEM,VALUE)
```

where

ITEM/I	Code to specify the process information to be retrieved. ITEM may be a field or a constant.
	`Code  Item`
	`1  User name`
	`4  Current Working Directory`

VALUE/An	Resulting value. VALUE must be an alpha field, large enough to store the requested data.
STAT/I	-4: VALUE is shorter than 15 bytes when asking for Process name, or shorter than 12 bytes when asking for User name. -3: VALUE is not alphanumeric -2: ITEM value is invalid (not 1 or 4) -1: Invalid number of arguments 1: OK, requested value returned Other: Error code returned

H.14.12 SETJPI - Setting/Changing Process Information

The SETJPI subroutine sets or changes certain process information, e.g. the current working directory. The general format is:

```
STAT = SETJPI(OP,ARG1)
```

where:

OP/I	Operation to perform. The following operation codes are defined: 1: Change Working Directory -1: Restore original working directory 2: Reload the TRANS_ENV file on the next branch even if the value of the TRANS_ENV statement or the TRANS_ENV logical name did not change.[a]
ARG1/An	For OP = 1 the absolute or relative path to the new working directory (may be a logical name that translates to the new working directory).
STAT/I	Return status. 1: OK -1: Invalid OP code -2: Argument 2 has wrong type -10: Cannot change to the specified drive letter -11: Cannot change to the specified directory

a.If the screen you branch to is in the same TRO you have to use the external branch syntax for this to work, i.e. if your TRO is XX.TRO and you are branching to the screen SCR2 in the same code, use:

2 XX/SCR2

to force a reopening of the TRO (as this is the only time a change in TRANS_ENV is checked).

H.14.13 SNDX - Calculate a Sound Index for a Name

The function SNDX is used to produce a 5 character soundex code from a last and a first name where names that **sound** alike have the same or similar soundex codes.

H.14.13.1 SNDX Syntax

```
INDEX = SNDX(LNAME,FNAME)
```

INDEX/A5	The first character is the first initial of the LNAME field, the next three digits are the numerical sound index of the LNAME field, and the last character is the first initial of the FNAME field.
LNAME/An	Field containing a person's last name.
FNAME/An	Field containing a person's first name.

H.14.13.2 SNDX Example

Given the following fields and values,

```
INDEX/A5
LNAME/A10 'GRIFFEL'
FNAME/A10 'DAVID'
```

then

```
INDEX = SNDX(LNAME,FNAME)
```

would result in the value "G614D" in the field "INDEX".

H.14.14 SPAWN - Create Subprocess from ADMINS Command

The ADMINS SPAWN subroutine spawns a subprocess to execute a host system command passed as an argument to the SPAWN subroutine.

The SPAWN subroutine can be told to locate and expand any logical names on the command line before spawning the command. This will be necessary if you spawn a non ADMINS command, and use a logical name in one or more of its command line arguments (because on Windows only ADMINS commands support logical names). This is done by calling SPAWN with the single argument COMMAND set to '%%+'. This will ensure that SPAWN will attempt to translate any possible logical names (i.e. words terminated by a colon (':')) in the spawned command. To tell SPAWN to stop translating logical names, call it with COMMAND set to '%%-'.

When turning translation on, the '%%+' may be followed with any number of valid termination characters for the translation. E.g. calling SPAWN with COMMAND = '%%+\:' will leave any translated value ending in '\' or ':' as is, and append '\' (the first of the termination characters) to any translated value that does not end in any of these characters.

The syntax of the subroutine is:[36]

```
STAT = SPAWN(COMMAND[,INPUT[,OUTPUT[,FLAGS[,PID]]]])
```

COMMAND/An(D) — Command to execute, e.g. 'CMP MYPROG'. May be an array to permit command arguments longer than 80 characters. If using array, it is developers responsibility to ensure unused array elements are blank.

FLAGS/I — Seven discrete values are used to designate optional behavior for the spawned process. These values can be summed to combine multiple attributes.

Value	Meaning
1	If set, the calling process continues to execute in parallel with the subprocess.
256	Run spawned process minimized. (Win32 only)
512	Run spawned process in Idle Priority Class. (Win32 only)
1024	Try to find the path to the application (1st word on the command line) in the registry. This makes it possible to launch certain installed applications (e.g. WINWORD) without the application's path being present in the PATH environment variable. If a path is not found, SPAWN tries to launch the application as is (Win32 only).

STAT/I — -1: Argument has invalid format
1: OK
Any other value: VMS error return status.

H.14.15 STACK - Store and Retrieve Data in a Stack

The stack subroutine allows the user to store and retrieve data[37] to and from a stack, in last-in first-out fashion.

STACK syntax:

STAT = STACK (OP,VALUE,CONTROL)

36. By default, TRANS refreshes the screen display upon return from a call to SPAWN. For compatibility with older applications, TRANS will ignore any simulated REF keystrokes that occur in the SETKEY buffer immediately after a call to SPAWN.
37. any ADMINS data type.

where::

OP/I — Operation: the type of operation to be performed. (Field name or constant).

Value	Meaning
1	Initialize the stack.
2	Move an element to the stack.
3	Retrieve an element from the stack.
4	Is the stack empty?
5	Is the stack full?
6	How many elements are currently on the stack?
7	Show the Nth element on the stack.

VALUE/__ — Field to operate on. (May be any data type)

CONTROL/I — Control Values (must be a field): used as an input or output argument, depending on the operation being performed, e.g. when the stack is to be initialized, CONTROL is used to specify the sizeof the stack. CONTROL use is summarized below:

Operation	Use
1	Input - Size of the stack
2	Output - 1 if full otherwise 0
3	Output - 1 if empty otherwise 0
4	Output - 1 if full otherwise 0
5	Output - 1 if empty otherwise 0
6	Output - Number of elements
7	Input - Element number

STAT/I Return Status:

Value	Meaning
1	Successful operation
-1	Illegal operation
-2	Stack size is less than one
-3	Memory allocation for the stack failed
-4	Stack was not initialized
-5	Type mismatch
-6	Element requested is out of range

1. Initialize (Operation 1): To use a stack you first have to initialize (create) the stack. For initialize operations, VALUE tells STACK the data type to be used, and CONTROL tells STACK the size (maximum number of elements) stack to create.
2. Move (Operation 2): To move data to the stack, VALUE contains the data to be moved. If the stack is full after the operation CONTROL is set to 1, otherwise CONTROL will be set to 0.[38]
3. Retrieve (Operation 3): When data is retrieved from the stack, VALUE will contain the last element that has been moved to the stack, i.e. the "top of the stack". If the stack is empty CONTROL is set to 1 otherwise it is set to 0.[39]
4. Is the stack empty? (Operation 4): If the stack is empty CONTROL is set to 1 otherwise it is set to 0.
5. Is the stack full? (Operation 5): If the stack is full CONTROL is set to 1 otherwise it is set to 0.
6. How many elements? (Operation 6): How many elements are currently on the stack? CONTROL is set to the number of elements in the stack.
7. Show Nth element (Operation 7): Show the Nth element on the stack. CONTROL tells STACK which element you want to look at.[40] VALUE will be set to the value contained in the Nth element in the stack. If N elements are on the stack, and you ask STACK to show an element number greater than N, STACK returns an error status.

The data on the stack is not stored in the DA array and can be used globally across screens.

38. If N elements are on the stack before the move operation, and the move operation was successful the stack will contain N + 1 elements after the operation.
39. If N elements are on the stack before the retrieve operation, and the retrieve operation was successful the stack will contain N - 1 elements after the operation.
40. You can think about the show operation as giving you the contents of the Nth element of an array consisting of all the elements in the stack. The show operation does not change the number of elements in the stack.

IMPORTANT! If the stack is not being used any more, i.e. before a branch to a screen that does not use the stack, it is good practice to "de-initialize" the stack, i.e. initialize the stack with CONTROL set to 0. This frees the memory allocated to the stack.

The following sample RMO statements use STACK to move repeated entries into the field ITEM onto a stack. The DUMPST subroutine loop is used to dump the entire stack into a local array.

```
LOCAL
.
.
.
STAT/I
INIT/I 1
PUSH/I 2
POP/I 3
SIZE/I 40
OK/I
INIFLG/A1 'Y'
XARRAY/A10(40)
ITEM/A10
PROGRAM
.
.
.
* If stack hasn't been initialized, initialize it.
*
IF INIFLG EQ 'Y' THEN ;
  STAT = STACK(INIT,ITEM,SIZE) ;
  IF STAT NE 1 THEN ERR = 201 ; STOP ; END ;
  INIFLG = 'N' END
*
* "Push" the contents of ITEM onto the stack.
*
STAT = STACK(PUSH,ITEM,OK) ;
  IF STAT NE 1 THEN ERR = 202 ; STOP ; END
.
.
.
* GOSUB routine to "pop" the entire stack into a
* local array.
*
DUMPST: ;
STAT = STACK(POP,TVAL,OK)
IF OK EQ 1 THEN RET END
XARRAY(J) = TVAL ; J = J + 1 ;
GOTO DUMPST
```

H.14.16 SUMMARY - REPORT TOTAL Style Summaries

The SUMMARY subroutine provides REPORT TOTAL style summaries on numeric data. The syntax is:

```
STAT = SUMMARY(FILE,KEYFLD,PERFLD,PERTYPE,
                          ACCFLDS,OPER,TARGET[,SELECT])
```

FILE/An — File name on which to operate.

KEYFIELDS/An(d) — An An array that contains the names of the fields containing the key values controlling the operation. E.g. if:

```
KEYFLD/A18(3) 'FY' 'ACCNT'  '  '
```

you receive accumulations for the FY Fiscal Year and ACCNT account.

You may also specify a range of key values. E.g.:

```
KEYFLDS/A18(5) 'FUND' 'DEP1' '-' 'FUND'
'DEP2'  '  '
```

would sum up all records with keys specified by the values in the fields FUND DEP1 and DEP2 (the presence of the single character '-' following DEP1 indicates that we are specifying the low and high values of a key range).

PERFLD/An — Name of period field. Must be a Date field, or a X9999 of the form YYPP (Year/Period) or an I field with the actual period, or if PERTYPE EQ 0 any field type (not used).

If PERFLD is defined as an array, e.g.

```
PERFLD/An(3)  'TRANSDATE'  'FSTDATE'
'LSTDATE'
```

TRANSDATE is the name of the field in FILE that contains the date or period. The RMO fields FSTDATE and LSTDATE contain the first and last date/period selected for accumulation.

If the value of PERTYPE is a year, at least a starting period is required to establish which year is year 1.

PERTYPE/I — How to calculate the accumulation periods:

0	No period (one sum for the whole range is provided)
1	Day (of month, 1-31)
2	Weekday
3	Week
4	Month
5	Quarter
6	Year

If the fiscal year does not start in January the starting month of the fiscal year can be signaled in the upper half of the PERTYPE field. This is done by taking the starting month, multiplying it by 256, and adding in the period type e.g.:

```
PERTYPE = (7 * 256) + 4
```

signals July as the start of the fiscal year, and to use month as the accumulation period (i.e. July is month one, June is month 12).

ACCFLDS/An(d) — An array with the names of the fields to accumulate.

OPER/A4(d) — An array of operators corresponding to each field. Valid operators are:

V	Accumulative values (the default, may be blank)
FI	First value
LA	Last value
MIN	The minimum value the field had
MAX	The maximum value the field had
AVG	The average value the field had
E	Number of non-zero values found

TARGET/An(d) — An array with the names of the target fields to accumulate into. Must be dimensioned for the maximum number of periods that can occur. The target arrays must be defined with the same data type and number of decimals as the ACCFLDS fields, except for OPER = E where the data type may be any of L, D or F (with or without decimals).

SELECT/An(d) — An optional An field or array containing an expression used to select records from FILE. If the select statement spans fields a ':' (colon) must be used as a continuation indicator (similar to SELECT statements spanning lines in DEFINE or REPORT). If the expression references a field in the virtual record (not in the FILE file) a '~' (tilde) must prefix the field name (like in TRANS Lookup), e.g.

```
TRANSDATE GT ~STARTDATE
```

where TRANSDATE would be in the FILENAME file, and STARTDATE would be in the RMOs virtual record.

STAT/I — Return status.

1: OK

0: No records found for supplied key value

-1: Invalid number of arguments

-2: Cannot open/find the file

-3: The period field supplied in PERFLD was not found

-4: Invalid period flag

-5: Period field has invalid type. Only I, DA, DT or X9999 supported.

-6: Number of fields to accumulate and number of target fields not the same.

-7: Invalid start of fiscal year (must be 1-12)

-8: Low/High period field not found

-9: Low/High period field not same type/length as period field

-10: Unable to establish starting year for period = year

-11: Error parsing select statement

-12: Error adding ~FIELD in select statement

-13: Error compiling select statement

-14: Too many accumulation fields (max:60)

-(100 + n): Key field n not found

-(200 + n): Key n does not exist, or wrong type or length

-(300 + n): Accumulator field n not found

-(400 + n): Accumulator field n not numeric

-(500 + n): Unknown operator for field n

-(600 + n): Target field n not found

-(700 + n): Target and source field n not same type or number of decimals.

-(800 + n): Period for field n greater than dimension of target field.

-(900 + n): Target field for operator E is not L, D or F.

E.g.

```
STAT =
SUMMARY(FILE,KEYFLDS,PERFLD,FLAG,ACCFLDS,
OPT,TARGET)
```

where:

```
FILE/A40 'ACCOUNTING:Transactions.mas'
KEYFLDS/A18(2) 'FY' 'ACCNT'
PERFLD/A18  'TRANSDATE'

FLAG/I   4
AFLDS/A18(3)  'DBAMT' 'CRAMT' 'DBAMT'
OPT/A4(4)     ' '     ' '     'MAX'
TARGET/A18(3) 'DBSUM ' 'CRSUM ' 'DBMAX '
DBSUM/D2(12)
CRSUM/D2(12)
DBMAX/D2(12)
FY/X9999 2001
ACCNT/A20 '01010051000'
```

Would open the file 'ACCOUNTING: Transactions.mas', use the values of the fields FY and ACCNT to access records in the file, use the field TRANSDATE to determine the month (FLAG=2), and accumulate the fields DBAMT and CRAMT into the arrays DBSUM and CRSUM, and the maximum debit amount per period in the DBMAX field.

H.14.17 SYNC - Synchronize Access to a File

Section 13.5 "SYNC - Synchronization Between ADMINS Commands" describes the SYNC command for synchronizing access to ADMINS data files or records, or other events, usually in command files. However, these ADMINS files can also be accessed via TRANS. The SYNC subroutine, callable from the RMO running behind the screen, provides this same capability in TRANS, so that events can be synchronized either with other TRANS users, or with command files.

NOTE

As discussed in Section 13.5 "SYNC - Synchronization Between ADMINS Commands", the logical name ADM$SYNC_HOLD must be assigned in the group or the system logical name table to point to the directory that contains the SYNCHOLD.EXE, e.g.

```
$ ASSIGN/SYSTEM DUA0:[ADMDIST.V32] ADM$SYNC_HOLD
```

in order to use the SYNC facilities

H.14.17.1 SYNC Syntax

```
STAT = SYNC(EF,ACTION,[LEVEL])
```

STAT/I	Status indicator. -1 means EF was not in the range of 50-59 0 means the flag (lock) referenced by an 'X' ACTION is not available. 2 means the flag (lock) referenced by an 'X' ACTION is available.
EF/I	Number of the flag involved (50-59).

ACTION/A1	Action to be taken. 'W' means to take flag EF and continue. The action of taking the flag makes the flag unavailable to another user. If flag EF is already unavailable (taken by another process), 'W' means wait until it is released, take it, and continue. 'S' means to release the flag EF thus making it available. 'X' returns the current setting of flag EF in STAT, 2 for available, 0 for not available.
LEVEL/I	Optional. Field or constant. If present and set to a value of 1, then the sync flag is in effect system-wide. Otherwise the flag is in effect for the group.

H.14.17.2 SYNC Example

Given the following fields and values,

```
STAT/I
EF/I 57
ACTION/A1 'W'
```

then

```
STAT = SYNC(EF,ACTION)
```

included in an RMO behind a screen would cause TRANS to check flag 57. If the flag is unavailable, then wait until it is available. When flag 57 is available, take the event flag and continue.

H.14.18 TTCOM - Communication With Another Terminal

The TTCOM subroutine is used to communicate with any device connected to a terminal interface line, e.g. another terminal, a printer, a cash register printer, or a micro computer. TTCOM must be able to allocate the device to prevent other users from accessing the device at the same time. Since the default terminal protection prohibits terminals from being allocated by an existing process, you must change the default before using TTCOM. TTCOM can directly deallocate the device, to free it for use by other users.

TTCOM provides two basic commands; WRITE characters to the device, and READ characters from the device. You may specify time-out values which are the number of seconds the read should wait before it returns to the calling process. You may also specify termination characters.

H.14.18.1 TTCOM Syntax

```
STAT = TTCOM(OP,DEVICE,STRING,PARM)
```

OP/I	=21: STRING/Ann contains character string to write to DEVICE. =22: STRING/I(n) contains integer values which should be printed as a character string to DEVICE. =23: Read DEVICE and return character string read in STRING/Ann =24: Read DEVICE and return character string read as integers in STRING/I =25: Deallocate DEVICE last used
DEVICE/A24	contains the name of the device to which you wish to communicate. It may be a logical name, but it must eventually translate to a physical device that is a terminal.
STRING/xx	For OP equal 21 or 23, STRING is defined as STRING/Ann, and for OP equal 22 or 24 it is defined as STRING/I(n). If OP=21, STRING/Ann contains an ADMINS string to be written to DEVICE, and if OP=22, STRING/I(n) contains n integer values to be interpreted as characters and written to DEVICE. If OP=23 or 24, STRING is a buffer to receive the character string read from device. If OP=23 it is returned as an ADMINS character string, and if OP=24 it is returned as integer values. For OP=25 STRING is a dummy variable.
PARM/I(5)	PARM contains parameters to the TTCOM subroutine.
PARM(1)	Length of STRING in number of characters. If write, number of characters to write from STRING. If read, number of characters read from DEVICE. If STRING is defined as /Ann, PARM(1) is number of bytes, if STRING is defined as /I, PARM(1) is number of words.
PARM(2)	If write specifies Carriage Control Character. FORTRAN Carriage control characters are used: HEX DEC Interpretation 20 32 Single-space 30 48 Double-space 31 49 Page eject 2B 43 Overprint 24 36 Prompt All other values are interpreted as single space CC characters. If read, specifies time-out value, i.e. the number of seconds the TTCOM read call should wait before it returns to the calling process.
PARM(3)	Ignored if write. If read, PARM(3)=1 signals that PARM(4) and PARM(5) contains a bit map to indicate termination characters.

PARM(4) PARM(5)	Bit map to indicate which characters should be interpreted as terminator characters for read (OP=23 or 24). The 16 bits in PARM(4) represents ASCII value 00 through 15, and PARM(5) represents ASCII value 16 through 31. If a bit is on, the corresponding character should be treated as a terminator.
STAT/I	Return status code from TTCOM 1 = OK 2 = OK, timeout (Read only) -1 = Device allocated to other user -2 = No such device -3 = Device mounted -4 = Invalid device -5 = Syntax or other error other positive value = VMS error code (decimal)

.

H.14.18.2 TTCOM Example

Assume a set of screens which uses a lot of Advanced Video functionality, and another set which uses other display techniques. When an option is chosen on a menu screen, the RMO behind the screen determines which family of screens to branch to. One call to TTCOM sends a request for device product identification code to the terminal, and a second call reads the answer from the terminal. The ANSI standard request for product identification code or request for device attributes is:

```
ESC   [   c
 27  91  99
```

and the answer from a VT100 terminal should be:

```
ESC   [   ?   1   ;      c
 27  91  63  49  59   ^  99
                      |
                      0 (48) = Base VT100, no options
                      1 (49) = Processor option (STP)
                      2 (50) = Advanced video option (AVO)
                      3 (51) = AVO and STP
                      4 (52) = Graphics option (GPO)
                      5 (53) = GPO and STP
                      6 (54) = GPO and AVO
                      7 (55) = GPO, STP and AVO
```

If the "option present" is 2, 3, 6 or 7 the terminal has Advanced Video.

Examine the following RMS:

```
LOCAL
.
DEVICE/A24 'SYS$COMMAND'
STAT/I
ASK/I(3) 27 91 99
ANSWER/I(20)
PARM/I(5)
.
PROGRAM
.
*
* Determine menu CHOICE
IF S$S EQ 'CHOICE' THEN ;
   IF CHOICE EQ 'R' THEN ;
```

```
     * Three integers to be sent to SYS$COMMAND
          PARM(1) = 3 ;
          STAT = TTCOM(22,DEVICE,ASK,PARM)
     * If error, get out
          IF STAT NE 1 THEN ;
             ERROR = STAT ; GOTO EOP END ;
     * Initialize parameters for answer, PARM(3) indicates
termination
     * characters bit map for PARAM(4) and PARAM(5)
          PARM(2) = 0 ;
          PARM(3) = 1 ;
          PARM(4) = 0 ; PARM(5) = 0 ;
          STAT = TTCOM(24,DEVICE,ANSWER,PARM) ;
     * If error, get out
          IF STAT LT 1 THEN ;
             ERROR = STAT ; GOTO EOP END ;
     * If the sixth value includes AVO option then branch to 'R'
screen
          IF ANSWER(6) EQ 50 OR 51 OR 54 OR 55 THEN ;
             B$B = 'R' ; GOTO EOP END ;
     * Otherwise branch to 'S' screen
          B$B = 'S' ; GOTO EOP END ;
     ...
```

H.14.18.3 TTCOM - New Operation Code

A new operation code added to TTCOM enables the setting and reading of communication parameters (baud rate, parity, etc.). The syntax is:

```
STAT = TTCOM (20,DEVICE,NULL,PARAM)
```

where:

- PARAM(1) is baudrate (e.g. 9600)
- PARAM(2) is Parity On or Off (1 is On, 0 is off)
- PARAM(3) is Parity setting. 0 = No parity, 1 = Odd parity, 2 = Even parity, 3 = Mark parity, 4 = Space parity
- PARAM(4) is Byte Size in bits (4-8)
- PARAM(5) is Number of Stop Bits (0=One, 1=1.5, 2=Two)

Any of the PARAM(n) values can be set to -1 (minus one) to leave the current setting intact.

A return value of -3 indicates that there was an error while setting the values.

On a successful return (STAT = 1) the PARAM(n) fields are filled in with the current settings from the device.

H.14.19 BLOBIO - Access Binary Large Object (BLOB) Field

BLOBIO allows binary large objects to be stored in and retrieved using the ADMINS BLOB field type. BLOB fields store binary large objects in ADMINS internal text files (.TCF/.TSF) in the same way that internal text (TI) fields store text documents (see Appendix K: "Using Text Fields"). The BLOB field makes the binary large object a part of the ADMINS data record, a field that can be stored and retrieved by key value, and remains associated with the ADMINS record as it moves around in sorts, moves etc., in exactly the same way as text documents stored in TI fields.

Be aware that, unlike internal text fields, ADMINS does not know what to do with a BLOB.[41] BLOBIO can only store or retrieve a BLOB, and put its content into a file or in memory. The developer must indicate in the RMO how the content is to be

handled, through a call either to the EXTERNAL subroutine (see Appendix H.14.8 "EXTERNAL - Call External Language Routine") or the SPAWN subroutine (see Appendix H.14.14 "SPAWN - Create Subprocess from ADMINS Command").

To incorporate a binary large object into an ADMINS data file include a field of data type BLOB in your .DEF file:

```
* EMPLOYEE.DEF
*      Employee File     *
*
MAS     1000
EMPNO   X999   KEY1    ! employee number
LNAME   A20            ! last
FNAME   A20            ! first
MIDDLE  A1             ! middle init.
IDPHOTO BLOB           ! photo
```

When this file is defined, IDPHOTO is used to store a pointer to the location of the object BLOBIO associates with this record.

BLOB fields cannot be edited with TRANS, and developers must make sure BLOB fields are never touched by the RMO.

The syntax of the BLOBIO subroutine is:

```
STAT = BLOBIO(FUNC,OPTION,BLOBFLD,WHERE,SIZE,FORMAT)
```

where:

FUNC/I	(field or constant) Tells BLOBIO what to do: 101: Write the BLOB into the file identified by WHERE. 102: Put the BLOB into memory at the address in WHERE. 201: Get the BLOB from the file identified in WHERE, and put it in the .tsf file. 202: Get the BLOB from the memory location in WHERE, and put it in the .tsf file. 900: Free memory at address in WHERE (no reading or writing).
OPTION/I	Option settings to modify the behavior of BLOBIO: 1: Allocate memory for FUNC = 102, and deallocate memory for function 202. 2: Delete file after it is written to the .TSF file (with FUNC = 201).
BLOBFLD/BLOB	Contains the ID of the binary large object, a pointer to its location in the .TCF/.TSFfiles. FUNC set to 201 and 202 loads BLOBFLD the first time BLOBIO is called for the object. MAKE SURE the BLOBFLD is written to disk after BLOBIO is called with FUNC set to 201 or 202.
WHERE/An	If you are reading or writing the BLOB from/to a file (FUNC = 101 or 201), this is an An field containing the name of the file.

41. The BLOBIO subroutine updates the virtual record in memory when it loads the BLOBFLD, but it does not write that record back to disk. In TRANS, for example, use W$W to force writing to the disk. Otherwise, the record may never be written to disk, and the pointer to the binary large object might be lost.

WHERE/F	If you are reading or writing the BLOB from/to a memory location (FUNC = 102 or 202), this is the address where the BLOB is located. This address may be obtained in two ways: 1) By calling EXTERNAL: have EXTERNAL allocate the memory, and return the address in the WHERE field. 2) For FUNC = 102, have BLOBIO allocate the memory by setting OPTION to 1. Normally, the function responsible for allocating the WHERE memory, should be responsible for freeing the memory. When reading or writong to a file (FUNC=101 or 201), BLOBIO supports the File Open Dialog Box (see Appendix H.12.1.1 "File Open Dialog Box" for more information).
SIZE/L	The size of the BLOB being handled. Required if the BLOB is in memory. If BLOB is in a file, SIZE is informational only, and is always set by BLOBIO.
FORMAT/I	(field or constant, must be greater than 256) A code to identify the format of the BLOB. (A BLOB might be a WORD or WordPerfect file, or a graphical image of some format, etc.) Stored with the BLOB when FUNC = 201 or 202, and retreived with it when FUNC = 101 or 102. FORMAT can be used to indicate which program is to be activated to handle the BLOB. We have left to developers how to define their own object handlers, thus making the object orientation of the BLOB data type open ended. Typically, BLOBIO would be called to get the binary large object, and then the RMO would activate the appropriate image to handle the BLOB depending on its FORMAT value.
STAT/I	`Return status from the BLOBIO subroutine:` 2 = Loaded OK, but unable to delete file (with OPTION=2) 1 = OK 0 = Nothing read/written -1 = Invalid function code -2 = BLOBFLD is not of type BLOB -3 = External file field not of type BLOB -4 = WHERE is not alphanumeric (An) for FUNC = 101 or 201 -5 = WHERE is not type F for FUNC = 102 or 202 -6 = SIZE not of type L -7 = Copy BLOBFLD to/from memory, and no size -8 = Trying to read nonexistent BLOB -9 = FORMAT not of type I -10 = Invalid FORMAT. Must be > 256 -11 = Open error on output file -12 = Open error on input file

The following RMS fragment shows the BLOBIO subroutine being used to access a word processor file stored in a BLOB.

```
*
*
*****************************************************
** If it already exists write the document to disk,  **
** use WORD PROCESSOR to edit (or create) it; write  **
** it back into ADMINS (TSF) file.                   **
** Delete word processing file.                      **
*****************************************************
*
IF M$M EQ 'UP' AND S$S EQ 'ILIN' AND ILIN EQ 'Y' THEN ;
   IF WPFILE GT 0 THEN ;          ! does BLOB exist?
      STAT = CRLOG(LOGNAM,FILE) ; ! write BLOB out to LOGNAM
      STAT = BLOBIO(101,OPTION,WPFILE,LOGNAM,SIZE,FORMAT)
   END ;
   STAT = SPAWN(WORDPROC) ;       ! spawn word processor
   OPTION = 2 ;                   ! set delete option
*                                 ! write back BLOB
   STAT = BLOBIO(201,OPTION,WPFILE,LOGNAM,SIZE,FORMAT) ;
*                                 ! clean up and
   FILE = ' ' ; ILIN = ' ' ;      ! delete logical
   STAT = CRLOG(LOGNAM,FILE) END
*
*
```

Appendix I:ADD: The ADMINS Data Dictionary

This document describes the purpose, syntax, and functionality of the ADMINS Data Dictionary (ADD), and illustrates its use in the development of a simplified, yet complete and integrated demonstration application. The reader is assumed to have general familiarity with the ADMINS software product, including both application development issues (i.e. what the commands do, syntax, etc.) and with using the various ADMINS commands to perform a task (i.e. special keystrokes, etc.).

I.1 Introduction

The ADMINS Data Dictionary system is based on an Entity/Relationship model, where the entities and relationships are tailored to the specific needs of the ADMINS programming environment. ADD provides a repository for information about the various entities and relationships that comprise an ADMINS-based information system. ADD enhances your ability to develop, maintain and document ADMINS-based applications in a consistent, organized, and integrated manner.

ADD is implemented via a family of TRANS screens that share a common layout and user interface. Figure I1-1 identifies the standard features of the ADD screen layout.

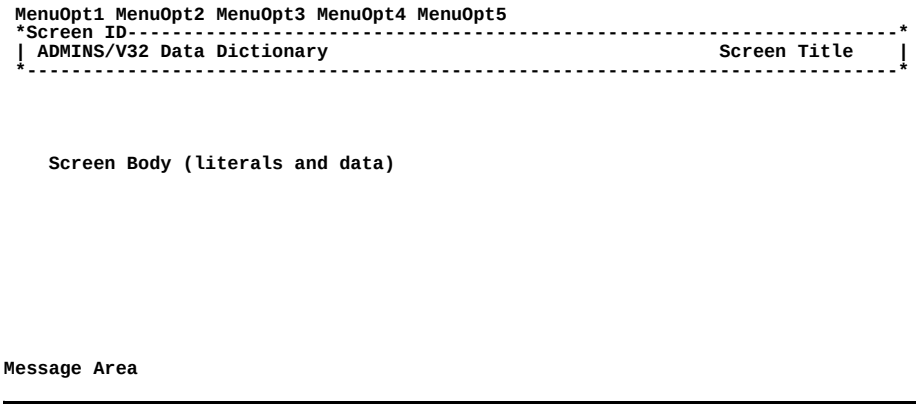


Figure I1-1 Generalized Screen Layout

The major options that are available from any screen panel are offered in the **menu bar** at the top of the screen. (The same choices will also normally be available through regular TRANS branch menus). In general, **HELP** is available throughout ADD, both at the screen level, and field-by-field. When appropriate, LOOKUP windows are available for displaying and selecting from the list of valid alternative responses.

I.1.1 Using the ADD Screens

All ADD activity begins with the Main Menu:

```
EXIT     HELP
*MENU-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                                 DICTIONARY MENU |
*-------------------------------------------------------------------------*
 Dictionary..: D1                 DEVELOPMENT DICTIONARY
                                  Current User: GINNY

  EL: Data Element Overview             CL: Codelist Repository Overview
  EA: Data Element Attribute            CR: Codelist Repository Attribute

  PE: Prototype Element Overview        CT: Codelist Table Overview
  PA: Prototype Element Attribute       CA: Codelist Table Attribute

  FI: File Overview                     DV: Data View Overview
  FA: File Attribute                    DA: Data View Attribute

  DR: Data Dictionary Reports           US: User Overview
  WH: Where Used Screen                 UA: User Attribute

 Your Choice.:               <Enter one of the listed 2 character codes>
```

Figure I1-2 The Main Menu

The ADD main menu selection alternatives direct you to overview and attribute screens for the various **entities** and **relationships** that make up the data dictionary.

Choose the screen you want by entering the two character code indicated on the menu.

The entity overview screens are gateways to families of screens that support all the query and maintenance activities for the corresponding entity type. The screen family for each entity type includes a screen for entering and altering the properties, or **attributes** of that entity. Screens are also provided for the specification of relationships (i.e. "file contains field" or "data view contains file element"), for text documentation of those relationships; and for codelist maintenance. Each of these activities is described in detail in later sections of this document.

I.1.2 Deleting Entities

Entities can be deleted by branching to the main attribute screen for that entity, locating the particular record, and then using TRANS DEL (ctrl/d) keystroke. **ADD will ignore the keystroke if any relationships for that entity to another entity remain at the time you attempt to delete it.**

Some examples:

For a **data element,** no prototype or codelist may be referenced in a record to be deleted, remove the prototype element or codelist reference before deleting the data element. A data element may not be deleted if it is included in any file relationships, remove the data element from the file relationship before deleting the data element.

No **codelist table** can be deleted while a prototype element or data element references that codelist table, delete the references to the codelist table from **all** the prototype or data elements before deleting the codelist table. No codelist table can be deleted while the codelist table values still exist, remove all the codelist table values before deleting the codelist.

I.1.3 The Demonstration Application: DEMO

Each step in the process of developing an ADD-based application will be illustrated utilizing examples taken from a simplified demonstration application: an order entry system for the New Tradition Bottling Co., an imaginary company engaged in bottling soft drinks. The completed data dictionary specification of the order entry system, and the demonstration application, are included with the ADMINS Data Dictionary distribution kit, for reference.

I.2 Edit Masks

An Edit Mask is a string of characters that specify editing and display properties for a field. Examples include:

- what type of characters are allowed in a certain position (numeric, alphabetic, alphanumeric, etc)
- what characters should be added for display purposes (e.g. parenthesis around the area code in a telephone number)
- providing default values

I.2.1 Edit Type

ADMINS recognizes the following Edit Types:

%E	Editable. The cursor always goes there.
%D	A default value is provided, but you may change it.
%H	Hard default. The cursor never stops at this position; the default value is inserted in the data.
%x	Where x can be any other character, e.g. **%(**, which inserts the x as a constant in the displayed value.

I.2.2 Edit Mask

Following the Edit Mask Type is a string of characters specifying allowable characters in each character position:

9	A numeric character (0-9)
*	An alphanumeric character (any character)
X	Alphanumeric, uppercase letter
x	Alphanumeric, lowercase letter
@	Alphabetic character (A-Z, a-z, plus local letters)
A	An Upper Case alphabetic character
a	A Lower Case alphabetic character

An edit mask for a US phone number could be:

```
%(%E999%)% %E999%-%E9999
```

which would case the character string *6174945100* to be displayed as (617) 494-5100 (only the numbers are actually stored in the data field).

I.2.3 Default Values

When the %D (soft default value) or %H (hard default value) is used as the Edit Type, the Edit Mask characters must be followed by a default value. An equal sign (=) indicates the end of the edit mask characters and the start of the corresponding default value. E.g. if you have a phone number field and most of the phone numbers you enter are in the 617 area code, you could use an edit mask like:

```
%(%D999=617%)% %E999%-%E9999
```

A default value of 617 is provided for the area code, but since the edit type is %D you can change it.

I.2.4 Examples

Assume you have an account structure of FUND, DEPARTMENT, OBJECT, PROJECT, which normally displays as 999-999-99999-9999. The general edit mask for this account structure would be:

```
%E999%-%E999%-%E99999%-%E9999
```

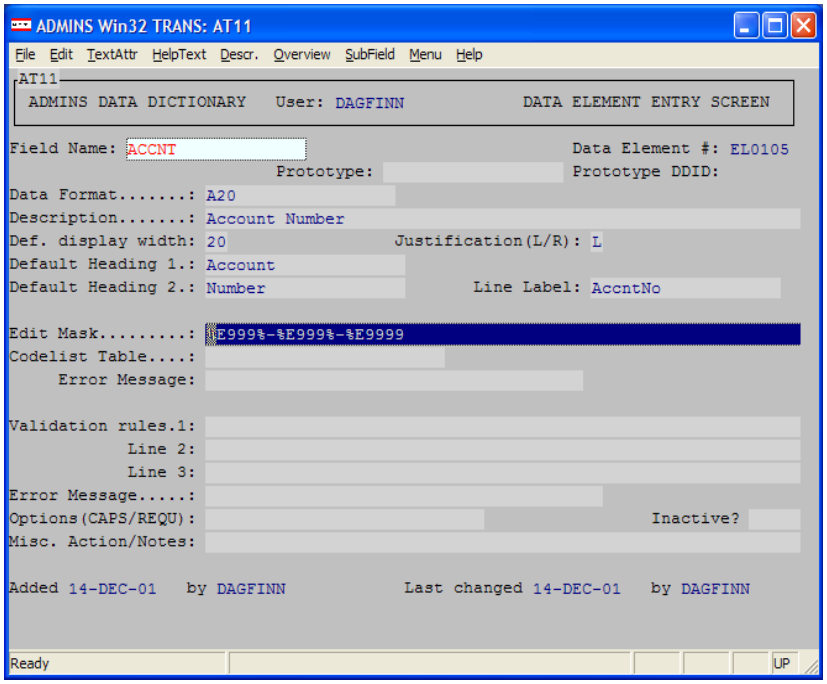
This is an edit mask where the cursor stops in every part of the account structure. Assume you are only allowed to enter accounts for department 201, and that your department never uses Project codes, the following edit mask would be adequate:

```
%E999%-%H999=201%-%E99999%-%H9999=0000
```

Edit masks are provided from the Data Dictionary, but should be able to be changed on the fly. If we use the same data entry screen to enter transactions for all kinds of accounts, the account type, to some degree, determines the edit mask (e.g. asset accounts do not use the department part, and a hard default of 000 is provided). For this purpose the EDITMASK subroutine is provided (see Appendix H.10.8 "EDITMASK" for more information).

I.2.5 Maintaining Edit Masks in the Data Dictionary

Edit Masks are initially specified for a field in the ADMINS Data Dictionary AT11 screen:



I.3 Subfields

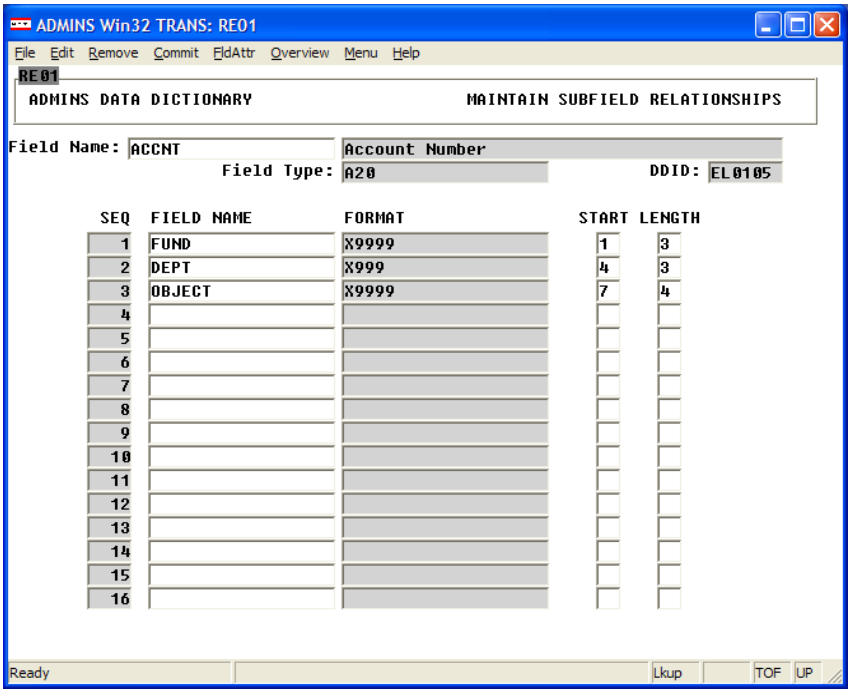
Another construct, that is separate from Edit Masks, but which more often than not is used together with Edit Masks, is subfields.

Consider the field ACCNT in the example above. According to the specified edit mask a 10 digit account would be displayed like e.g.:

100-110-5100

In fund accounting the three first digits would be the FUND number, the three next digits the DEPT (department) number, and the last four digits the OBJECT number (the order of these components may of course vary in different organizations).

To be able to easily have access to each of these sub-accounts we define (in the Data Dictionary) three fields, FUND/X999, DEPT/X999 and OBJECT/X9999 and then, under the ACCNT field, define these three fields as sub-fields of the ACCNT field:



In the FIELD NAME fields, we type (or look up) the names of the fields we want as sub-fields, and in the START and LENGTH columns we specify where each sub-field starts, and its length, in the mother field.

Now, if the field ACCNT is used in a file, the fields ACCNT.FUND, ACCNT.DEPT and ACCNT.OBJECT would be available in e.g. screens, RMOs and reports accessing that file.

I.4 Data Elements

Data elements are what ADMINS developers and users have traditionally called data **fields.** They are the basic building blocks of any ADMINS information processing system. With the ADMINS Data Dictionary you can specify all the general properties of the application's data elements in one place.

I.4.1 Data Elements Overview Screen

The Data Elements Overview screen presents a listing of the data elements that have been entered[1] in the data dictionary:

```
ATTRIB   LOCATE   DEF      HELP     MENU
*AT00-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                          ENTITY OVERVIEW SCREEN |
*-------------------------------------------------------------------------*

 FIELD NAME              DESCRIPTION
 CUSTID                  Customer Identification Code
 CUSTOMER                Customer Name
 DADDR1                  Delivery contact address line
 DADDR2                  Delivery contact address line
 DCITY                   Delivery contact address (City)
 DCONTACT                Delivery contact name
 DELIVNOT                Delivery Notes
 DPHONE                  Delivery contact telephone number
 ORDER#                  Order Number
 PAGE                    Page Number (Order)
 SADDR1                  Sales contact address line
 SADDR2                  Sales contact address line
 SALESREP                Sales Representative
 SCITY                   Sales contact address (City)
 SCONTACT                Sales contact name
```

Figure I2-1 Data Elements Overview

If the entity you want is not displayed, search for it by selecting the LOCATE function. You will be prompted for the entity name to search for. Or you can browse through the list using the various TRANS keystrokes for moving from record to record (i.e. Prev Screen, Next Screen, Ctrl/n etc.).

To inspect or alter the attributes for one of the elements, move the cursor to that element, and branch to the attributes screen (via the menu bar is easiest!)

1. If no elements have been previously entered in the Data Dictionary, the Dictionary's internal elements will display. Once a data element is entered these internal elements are not displayed.

I.4.2 Data Elements Attributes Screen

Use the Data Elements Attributes screens to specify the data elements for your applications.

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: SCONTACT           DD_User: GINNY            Data Element #: EL0109
                             Prototype: NAME             Prototype DDID: PE0101
Data Format.......: A60
Description.......: Sales contact name
Def. display width: 60
Justification(L/R): L
Default Heading 1.: Contact                    Line Label: SContact
Default Heading 2.: Name

Codelist Table....:
     Error Message:

Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):                                                  Inactive?
Misc. Action/Notes:


Added 15-JAN-03   by GINNY             Last changed 15-JAN-03   by GINNY
```

Figure I2-2 Data Elements Attributes Screen

For each element the following attributes are displayed (Help is available on a field by field basis.)

Field Name:	Any valid ADMINS field name.
DD_User:	User name of current DD_User (internally generated).
Data Element #:	An internally generated and used Data Dictionary ID number for the field.
Prototype:	The name of the prototype element this field is to be based on (see Appendix I.5 "Prototype Data Elements"). If a prototype element is specified for a data element ADD will immediately assign to that data element all the attributes of the prototype. You may then override any of these default attributes.
Prototype DDID:	The Data Dictionary ID number of the prototype element (internally generated).
Data Format:	A valid ADMINS data type.
Description	Up to 60 characters available to describe the field.
Default display width:	Default width for GENED and automatically formatted REPORTs. If not entered the maximum display width for the ADMINS data type will be used[a].
Justification:	L(eft) or R(ight). If not entered alpha fields will be left justified and numeric fields right justifiedSection "*".
Default Heading:	Two lines of default heading (20 characters each). Will be used for multi-colum displays (e.g. GENED, automatically formatted REPORTs).
Line Label:	Label to be used instead of field name, in single-record GENED screens[b].
Codelist Table:	The name of the associated codelist table for verifying input values.
Error Message:	Allows the developer to implement a specific error message to be displayed when an value entered into a field validated against a codelist table is not present in the codelist table. If this field is left blank the default message: `not present in Codelist Table` is displayed.
Validation Rule:	Up to three lines of logic to verify entered values. Use same syntax as ADMINS SELECT, except that the data element name is implied at the start of the statement, e.g. "GT 0", "BET 0 AND 1000".[c]
Error Message:	Message to display if entry does not pass validation rule.
Options:	CAPS, REQUIRE, etc.

Inactive?	You can enter "Y" in the Inactive field. This has the effect that even if the field is present in screens it is faded into the background making it obvious that this is not an active field. The main purpose of this feature is to leave fields that are not in use by a certain installation in the screen, but automatically disabling the field, thus cutting down on necessary customization of screens etc.
Misc. Action/Notes	Field without pre-defined Data Dictionary purpose. Available to store information for any application-specific purpose. This information can be accessed in applications using the EL_MISC attribute mnemonic.
Custom Fields 1, 2, 3	Fields without pre-defined purpose, can be used for any application-specific purpose.

a.Width and justification can also be accessed for use in screens. There are two techniques: first, via the W% and J% tokens when using “precise placement” syntax as described in Section 5.6.1 ″Precise Placement of Fields″, and second, via the special <W%FIELDNAME> and <J%FIELDNAME> parameters decribed in Section 5.17.2 ″Data DIctionary Parameters″.
b.Can also be used to supply a label for a field in TRANS, either via the special field L%FIELD, described in Section 5.5.8.11 ″L%FIELD: Get Label for field from data dictionary″ or by the special parameter <L%FIELD>, described in Section 5.17.2 ″Data DIctionary Parameters″.
c.A special syntax is provided to handle complex validation logic. For example, to indicate that codes between 20 and 29 or 40 and 49 are valid, enter: "BET 20 AND 29 OR (<VALUE> BET 40 and 49)" At compilation (SCREEN and TRANS GENED) and at data entry the special constant <VALUE> is replaced with the actual name of the field being validated. This implementation allows the complex validation logic to be applied to a prototype element, where the name of the field that references the prototype element, rather than the name of the prototype element, needs to be substituted into the statement.

ADD maintains control information for each entity (displayed at the bottom of the screen):

Added:	The date when this entity was entered.
by:	The user name when this entity was entered into the dictionary.
Last changed:	The date when this entity was last altered.
by:	The user name when this entity was last altered.

Via the menu bar you may return to the main menu, or to the data element overview screen. Additional menu bar alternatives branch to screens for entering and updating the help text to be associated with this data element, for entering and updating the text attributes of this data element, and for entering and updating descriptive text (documentation) for this data element.

I.4.3 Text Fields

If the **Data Format** attribute for a data element is specified as either **TInn** or **TXnn,** i.e. the internal or external text data types, you must specify the **Text Field Attributes** for that data element. Text field attributes are specified in the **Text Element Default Attributes** screen.

```
ELEMENT  HELP     MENU
*AT12-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                 TEXT ELEMENT DEFAULT ATTRIBUTES  |
*-------------------------------------------------------------------------*
Element Name: DELIVNOT            DD Id #: EL0140      DD_User: GINNY

Data Format.........: TI60
Description.........: Delivery Notes

Default Ruler......:         2  L-------T--------T--------T-------T-------T-----
Maximum Line Length:        60
Initialization File:
```

Figure I2-3 Text Element Default Attributes Screen

The Text Element Default Attributes screen first displays the **Element Name, DD ID#, DD_User, Data Format,** and **Description** from the Data Element attributes record for the text field.

The attributes that are specified in this screen are as follows:

Default Ruler

Specifies the **text ruler** that will automatically be inserted at the beginning of this text field when it is first entered or loaded. Use FIND to look up the text rulers defined in the Dictionary, and SELECT the one you want.

To define new rulers in the Data Dictionary, to remove rulers, or to alter existing rulers, you must use the Update Internal Codelist Tables screen, described in Appendix I.7.2.1 "Update Internal Codelist Tables ", to alter codelist table ADM$DD_TEXT_RULERS (DD_ID CT0012).[a]

Maximum Line Length

Maximum length of a **text ruler** that can be used with this field.

Initialization File

Specifies the initialization file that will be used for this text field when it is first entered or loaded. Use FIND to look up the initialization files that are defined in the Dictionary, and SELECT the one you want.

To define new initialization files in the Data Dictionary, or to remove or change initialization files, you must use the **Update Internal Codelist Tables** screen, described in Appendix I.7.2.1 "Update Internal Codelist Tables ", to alter codelist table **ADM$DD_TEXT_INITFILES** (DD_ID CT0013).[b]

a. See Appendix J.3 "Rulers" for an explanation of the syntax for specifying rulers.
b. See Appendix J.8 "The Text Initialization File" for an explanation of initialization files.

I.4.4 Help Text

Use the data element Help text screen to enter or view the information you want to be displayed when a user asks for "help" (e.g. presses the HELP key) for a field in a TRANS application.

If no "help" has been previously entered for the data element, ADD asks whether you wish to enter any:

```
No Help Text available. Do you want to add Help Text?
```

If you respond "y" ADD will call a text editor for you to use to enter the help text. You may enter up to 30 lines of up to 76 characters each. When you are finished entering the help text, exit the editor, and your new entry will be shown in the text display window. (You may use the menu bar NEXT_PG and PREV_PG options to browse around in the text if it exceeds the size of the window.)

As a time-saving alternative to entering new help text from scratch, you may **copy** help or descriptive text from another entity. This allows you to re-use help or descriptive text already written for other entities, perhaps changing it slightly for this new use.

To use copy:

1. Reply "N" to

   ```
   No Help Text available. Do you want to add Help Text?
   ```

2. Choose the COPY option on the menu bar. ADD will branch to the **Descriptive/ Help Text Copy screen:**

```
RETURN   FIND     COPY     HELP     MENU
*-CP00-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY             FIELD NAME DESCRIPTIVE/HELP TEXT COPY|
*-------------------------------------------------------------------------------*

Receiving Entity: DELIVNOT                ( EL0140  )
Copied Entity...:                         (         )

 *---------------------------------------------*-------------------------------*
 |                                             |Type Name_____________________ |
 |                                             |EL   7QTY                      |
 |                                             |EL   CUSTID                    |
 |                                             |EL   CUSTOMER                  |
 |                                             |EL   DADDR1                    |
 |                                             |EL   DADDR2                    |
 |                                             |EL   DCITY                     |
 |                                             |EL   DCONTACT                  |
 |                                             |EL   DELIVNOT                  |
 |                                             |EL   DPHONE                    |
 *---------------------------------------------|EL   ORDER#                    |
                                               *-------------------------------*
```

Figure I2-4 Descriptive/Help Text Copy Screen:

3. Press the FIND key. ADD displays a list of the entities currently defined in the Data Dictionary. Use SELECT to indicate your candidate entity to provide the text to copied.
4. Use Menu Bar option FIND to indicate whether Help text or Descriptive text is to be reviewed. ADD will then display the indicated text.
5. If you decide to copy the displayed piece, use Menu Bar option COPY. Otherwise, you can either return to the help or descriptive text screen you came from, or you can use this screen to review help or descriptive text for other entities (items 3 and 4 above)

Menu Bar option **COPY** copies the selected text piece to the "receiving entity".

To alter the help text for a data element, choose the EDIT option on the menu bar. ADD will call a text editor to edit the current version of the help text. When you are finished editing the text, it will be shown in the text display window.

When the user asks for help, up to 18 lines of this text will be displayed at the bottom of their screen. If you provide more than 18 lines of help, the user will be able to browse back and forth in the text using the Prev Screen and Next Screen keys. **This Data Dictionary HELP text will be displayed only if no HELP has been provided for the field otherwise** via traditional ADMINS methods (e.g. in a specific HELP file for the screen).

I.4.5 Descriptive Text for Application Documentation

ADD provides an integrated documentation maintenance facility that is implemented in the same way as the User Help Text facility. Use the data element descriptive text screen to enter or view documentation for data elements.

If no documentation has been previously entered for the data element, ADD asks whether you wish to enter any:

```
No description available.
```

```
Do you want to add descriptive text?
```

If you respond "y" ADD will call a text editor for you to use to enter the descriptive text. Up to 40 lines of up to 60 characters each may be entered. When you are finished exit the editor; your new entry will be shown in the text display window. (You may use the menu bar NEXT_PG and PREV_PG options to browse around in the text if it exceeds the size of the window.)

As a time-saving alternative to entering new descriptive text from scratch, you may **copy** help or descriptive text from another entity. This allows you to re-use help or descriptive text already written for other entities, perhaps changing it slightly for this new use. See the description in Appendix I.4.4 "Help Text" for how to utilize Menu Bar option **COPY.**

To alter the text description for a data element, choose the EDIT option on the menu bar. ADD will call a text editor to edit the current version of the text. When you are finished editing the new version will be shown in the text display window.

I.4.6 DEMO: Describing Data Elements

Two basic requirements of the NTB order entry system will be tracking information about customers, and information about their orders for NTB's products. To begin, we'll describe four data elements:

CUSTID	the customer identification number
CUSTOMER	the customer name
ORDER#	the order number
DELIVNOT	a text field that contains delivery information for the customer.

In the ADD main menu enter "EA" to branch to the Data Element Attribute Screen.

In the attributes screen the four records, CUSTID, CUSTOMER, and ORDER# are inserted as follows:

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: CUSTID             DD_User: GINNY            Data Element #: EL0101
                             Prototype:                  Prototype DDID:
Data Format.......: XA9999
Description.......: Customer Identification Code
Def. display width:  5
Justification(L/R): L
Default Heading 1.: Cust                       Line Label: CustID
Default Heading 2.: ID

Codelist Table....:
     Error Message:

Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):


Added 24-AUG-90   by GINNY             Last changed 06-JUN-91   by GINNY
```

Figure I2-5 Data Element CUSTID

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: CUSTOMER           DD_User: GINNY            Data Element #: EL0102
                             Prototype:                  Prototype DDID:
Data Format.......: A60
Description.......: Customer Name
Def. display width: 60
Justification(L/R): L
Default Heading 1.: Customer                   Line Label: Customer
Default Heading 2.: Name

Codelist Table....:
     Error Message:

Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):


Added 24-AUG-90   by GINNY             Last changed 24-AUG-90   by GINNY
```

Figure I2-6 Data Element CUSTOMER

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: ORDER#             DD_User: GINNY            Data Element #: EL0103
                            Prototype: NAME         Prototype DDID:

Data Format.......: X999999
Description.......: Order Number
Def. display width:  6
Justification(L/R): L
Default Heading 1.: Order                      Line Label: Order#
Default Heading 2.: Number

Codelist Table....:
     Error Message:

Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):

Added 24-AUG-90   by GINNY              Last changed 13-JUN-91   by GINNY
```

Figure I2-7 Data Element ORDER#

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: DELIVNOT           DD_User: GINNY            Data Element #: EL0140
                            Prototype:                Prototype DDID:
Data Format.......: TI60
Description.......: Delivery Notes
Def. display width: 60
Justification(L/R): L
Default Heading 1.: Delivery                   Line Label: Deliv. Notes
Default Heading 2.: Notes

Codelist Table....:
     Error Message:

Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):

Added 10-JUN-91   by GINNY              Last changed 10-JUN-91   by GINNY
```

Figure I2-8 Data Element DELIVNOT

DELIVNOT is a text field, so we now need to specify its text attributes.

Choose Menu option **TEXTATTR** to branch to the **Text Element Default Attributes** screen. The text attributes for DELIVNOT are inserted as follows:

```
ELEMENT  HELP     MENU
*AT12-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                 TEXT ELEMENT DEFAULT ATTRIBUTES  |
*-------------------------------------------------------------------------*
Element Name: DELIVNOT            DD Id #: EL0140      DD_User: GINNY

Data Format........: TI60
Description........: Delivery Notes

Default Ruler......:         2  L-------T--------T--------T-------T-------T-----
Maximum Line Length:        60
Initialization File:
```

Figure I2-9 Default Text Attributes for DELIVNOT

We'll need to track many different items for both customers and their orders. For customers, we'll need address, phone number, contact name, etc. For orders, we need to track who the customer is, who the salesperson is, how many units of which products are ordered, etc.

We could start right now to enter the names and addresses we'll need to keep track of for the customer. We could specify a sales contact name field. We could specify a delivery contact name field. We could specify fields for the phone numbers of these two people, and possibly for different addresses also. We would have to re-describe all the attributes of these sets of data elements that are really the same thing **used** in different ways. And worse, if we wanted to alter one of these common attributes, we would have to change that attribute in every instance.

Appendix I.5 “Prototype Data Elements” of this document will introduce prototype elements, which will save us most of the trouble of re-specifying these attributes for every data element in a group.

I.4.7 DEMO: Entering User Help

To enter default help text for the CUSTID element, go to the data element attributes screen for CUSTID, then choose menu bar option **HELPTEXT.**

Because no "help" has been entered for the CUSTID, ADD asks whether you wish to enter any:

```
No Help Text available. Do you want to add Help Text?
```

Reply "Y" to enter User Help. Figure I2-10 below shows an in-progress editor session where User Help for CUSTID is being developed.

```
Customer Identification Code:

Enter the Customer Identification Code (CustID).
If you do not know the CustID press the "Find"
key, which will display a Lookup window of all
the valid Customers.  In the Lookup window press
the "Select" key to enter the CustID of the
highlighted entry.









0   TRANS.TXT        Ln=1     Pg=1:3      Help=HELP    MAIN    WRAP TXT     INS
```

Figure I2-10 Entering User Help for Data Element CUSTID

Figure I2-11 below shows the completed User Help entry for CUSTID.

```
NEXT_PG  PREV_PG  EDIT     COPY     ELEMENT  OVERVIEW HELP     MENU
*AT16-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                          DATA ELEMENT HELP TEXT |
*-------------------------------------------------------------------------------*

EL0101  CUSTID              XA9999                          1
        Customer Identification Code
 *-------------------------------------------------------------------------*
 |Customer Identification Code:                                            |
 |                                                                         |
 |Enter the Customer Identification Code (CustID).                         |
 |If you do not know the CustID press the "Find"                           |
 |key, which will display a Lookup window of all                           |
 |the valid Customers.  In the Lookup window press                         |
 |the "Select" key to enter the CustID of the                              |
 |highlighted entry.                                                       |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 |                                                                         |
 *-------------------------------------------------------------------------*
```

Figure I2-11 User Help for Data Element CUSTID

To re-edit the User Help text for a data element, choose the **EDIT** option on the menu bar of the Data Element Help Text screen.

I.4.8 DEMO: Documentation

While in the Data Elements Attributes screen, choose the **DESCR.** option in the menu bar to enter documentation for the CUSTID field. ADD will prompt:

```
No description available.
Do you want to add descriptive text?
```

Reply "Y" to enter documentation for CUSTID (ADD starts an editor session).

Figure I2-12 below shows the completed Documentation entry for CUSTID.

```
        |                                                          |
        *----------------------------------------------------------*



NEXT_PG  PREV_PG  EDIT     ELEMENT  OVERVIEW HELP     MENU
*AT19-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                    DATA ELEMENT DESCRIPTIVE TEXT  |
*-------------------------------------------------------------------------------*

EL0101  CUSTID              XA9999                          1
         Customer Identification Code
        *-----------------------------------------------------------*
        |The Customer Identification Code is used                   |
        |thoughout the New Tradition Bottling Co.                   |
        |System.  All Customers will be referenced                  |
        |via their Customer Identification Codes.                   |
        |                                                           |
        |                                                           |
        |                                                           |
        |                                                           |
        |                                                           |
        |                                                           |
        |                                                           |
```

Figure I2-12 Documentation for Data Element CUSTID

I.5 Prototype Data Elements

Prototype data elements describe a set of attributes that are shared by a group or class of data elements in an application. Defining a prototype saves the trouble of re-specifying these attributes for every data element in the group. For example, you may need to specify telephone numbers in several different contexts in an application: home phone, business phone, telephone number of person to contact in an emergency, etc. Although used for entirely different purposes **each instance shares some standard attributes.** We can specify a prototype element, PHONE, that would allow us to set out these standard attributes, that we would expect to occur whenever a phone number is used in the application. We could expect all instances have 10 digits: a three digit "area code", followed by a three digit local exchange, followed by four digits. We could decide that all phone numbers should be labeled "Phone#", have a column heading of "Telephone" and a display width of 10, etc.

Then, in the ADD data element entry screen, when we specify the data element for each specific use, e.g. "Home telephone number", we can reference the prototype element for phone numbers. ADD will immediately assign to that data element all the attributes of the prototype.

I.5.1 Prototype Elements Screen Family

The Prototype Elements screen family is identical in structure to the Data Elements screen family described previously. The overview screen lists all the prototype elements that have been specified. The attributes screen allows you to enter and alter the attributes of the prototype elements. Screens are also provided for entering and updating help text,[2] entering and updating descriptive text for documentation purposes, and for specifying the text attributes of text fields.

I.5.2 Relationship of Prototype Elements to Data Elements

As soon as you relate a data element to a prototype element in the data element attributes screen (see Appendix I.4 "Data Elements"), ADD "fills in" all the attributes of that data element with values taken from the prototype element. In fact, the entries for all the attributes of that data element are loaded with a pointer, or token, that tells ADD to refer to the prototype element for each attribute.

All the attributes of the prototype element, **except the data format attribute,**[3] may be overridden for the data element. To override the "default" prototype value for an attribute of the data element, enter a new value for that attribute in the data element screen. What you type in will **replace the pointer to the prototype value** for that attribute, terminating the connection to the prototype for that attribute. **All the other attributes remain tied to the prototype,** unless and until they are also overridden. This is an important point because **changing an attribute in a prototype element changes every data element attribute that remains tied to it.**

Returning to the telephone number example used at the start of this section, if we specify "Telephone" as the column heading for the prototype element PHONE, and we relate the data element HOMEPHONE to the prototype element PHONE, "Telephone" becomes the column heading for HOMEPHONE. Later on, if we were to change the column heading attribute of the **prototype element** PHONE, to "Phone No." the column heading attribute of the **data element** HOMEPHONE would also change to "Phone No.".

On the other hand, if we had overridden the prototype attribute by entering "Home Phone" as the column heading for data element HOMEPHONE, then when the column heading attribute of the prototype PHONE is subsequently changed the column heading attribute for HOMEPHONE would not be changed, the connection to the prototype having been broken for the column heading attribute.

2. Help text specified for prototype elements will be displayed only if no HELP has been otherwise provided for the field either via traditional ADMINS methods (e.g. in a specific HELP file for the screen), or via help text specified for the data element itself.
3. The data format taken from a prototype element cannot be overridden. Any attempt to change the data format of an element that references a prototype element will result in an error message.

I.5.3 DEMO: Describing Prototype Elements

We have already specified two of the data elements that we need to track for each of NTB's customers, CUSTID, the customer ID, and CUSTOMER, the customer name. We can now define four **prototype elements** that we will be able to refer to multiple times in describing customers for the NTB order entry system:

NAME	A person's name
PHONE	A telephone number.
ADDRESS	One of the lines in an address.
CITY	City in an address.

In the ADD main menu, enter "PA" to branch to the Prototype Element Attributes screen.

Records for these four prototype data elements are inserted as follows:

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT01-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                  PROTOTYPE ELEMENT ENTRY SCREEN |
*-------------------------------------------------------------------------*
Prototype Field Name: NAME              DD_ID#: PE0101 DD_User: GINNY

Data Format.......: A60
Description.......: contact name
Def. display width: 60
Justification(L/R): L
Default Heading 1.: Contact                    Line Label: Contact
Default Heading 2.: Name

Codelist Table....:
     Error Message:

Validation rules..:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):

Added 24-AUG-90   by GINNY             Last changed 24-AUG-90   by GINNY
```

Figure I3-1 Prototype Element NAME

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT01-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                  PROTOTYPE ELEMENT ENTRY SCREEN |
*-------------------------------------------------------------------------*
Prototype Field Name: PHONE             DD_ID#: PE0104 DD_User: GINNY

Data Format.......: A14
Description.......: contact telephone number
Def. display width: 14
Justification(L/R): R
Default Heading 1.: Telephone                   Line Label: Phone
Default Heading 2.: Number

Codelist Table....:
     Error Message:

Validation rules..:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):

Added 24-AUG-90   by GINNY              Last changed 24-AUG-90   by GINNY
```

Figure I3-2 Prototype Element PHONE

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT01-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                  PROTOTYPE ELEMENT ENTRY SCREEN |
*-------------------------------------------------------------------------*
Prototype Field Name: ADDRESS           DD_ID#: PE0102 DD_User: GINNY

Data Format.......: A60
Description.......: Contact address line
Def. display width: 60
Justification(L/R): L
Default Heading 1.: Address                     Line Label: Address
Default Heading 2.:

Codelist Table....:
     Error Message:

Validation rules..:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):

Added 24-AUG-90   by GINNY              Last changed 24-AUG-90   by GINNY
```

Figure I3-3 Prototype Element ADDRESS

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT01-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                    PROTOTYPE ELEMENT ENTRY SCREEN |
*-------------------------------------------------------------------------*
Prototype Field Name: CITY              DD_ID#: PE0103 DD_User: GINNY

Data Format.......: A40
Description.......: contact address (City)
Def. display width: 40
Justification(L/R): L
Default Heading 1.: City                        Line Label: City
Default Heading 2.:

Codelist Table....:

Validation rules..:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):


Added 14-AUG-90   by GINNY             Last changed 14-AUG-90   by GINNY
```

Figure I3-4 Prototype Element CITY

I.5.4 DEMO: Relating Elements to Prototypes

Using the prototypes we described in the previous subsection, our job of describing some of the data elements we need becomes much easier. By referring to the prototype elements indicated, we now can specify the following data elements:

Data Element	Refer to Prototype	Description
DCONTACT	NAME	Delivery contact person
DPHONE	PHONE	Delivery contact phone
DADDR1	ADDRESS	Delivery address (line 1)
DADDR2	ADDRESS	Delivery address (line 2)
DCITY	CITY	Delivery address city
SCONTACT	NAME	Sales contact person
SPHONE	PHONE	Sales contact phone
SADDR1	ADDRESS	Sales address (line 1)
SADDR2	ADDRESS	Sales address (line 2)
SCITY	CITY	Sales address city

The two screens below illustrate the entries for DCONTACT and SCONTACT.

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: DCONTACT           DD_User: GINNY            Data Element #: EL0104
                             Prototype: NAME             Prototype DDID: PE0101
Data Format.......: A60
Description.......: Delivery contact name
Def. display width: 60
Justification(L/R): L
Default Heading 1.: Contact                    Line Label: DContact
Default Heading 2.: Name

Codelist Table....:
     Error Message:

Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):


Added 24-AUG-90   by GINNY             Last changed 24-AUG-90   by GINNY
```

Figure I3-5 Data Element DCONTACT

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: SCONTACT           DD_User: GINNY            Data Element #: EL0109
                             Prototype: NAME             Prototype DDID: PE0101
Data Format.......: A60
Description.......: Sales contact name
Def. display width: 60
Justification(L/R): L
Default Heading 1.: Contact                    Line Label: SContact
Default Heading 2.: Name

Codelist Table....:
     Error Message:

Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):


Added 24-AUG-90   by GINNY             Last changed 24-AUG-90   by GINNY
```

Figure I3-6 Data Element SCONTACT

I.6 Files

In ADMINS data elements (fields) are normally assembled into **data files.** Prior to the implementation of ADD, specification of the structure and other attributes of a file would be accomplished by creating a file definition instruction file (a "DEF") to be read by the ADMINS DEFINE command. With ADD, files are specified using the FILES screen family.

As with all ADD screen families, the **File Overview** screen is the entry screen for the family, and displays a list of all the files that have been identified. The **file attributes** screen is used to describe the major file attributes. The **maintain file contains element relationship** screen is used to specify which elements (fields) belong to a file, and how those elements are used, i.e. to enter the "DEF" of a file. The **data file descriptive text** screen is used to enter and update text documenting the purpose of the file.

I.6.1 File Attributes screen

For each file the following attributes are displayed:

File Name	Must be unique within the current dictionary. Must follow VAX/VMS rules for the first part of a file name (the second part of the file name, "File Type", is described below). Together File Name and File Type will become the name of an ADMINS data file when it is created on disk.
Description	Up to 60 characters that describe the purpose of the file
Directory	The physical directory, or a logical name that points to the physical directory, where the ADMINS data file will be located.
File Type	The file type, or file extension, for the data file (e.g. MAS, IDX, DER, etc.)
# of Records	The number of records the file should be defined for initially.
# of Log Records	Number of records the field log file should be defined for initially (Optional). If not entered no field log file is created when the file itself is created.
Index Only File (Y)	Provides option for DEFining a file /IXONLY (index-only). This option can only be used when all the fields are key fields. See Section 2.2.5 "IXONLY qualifier: Create Index-only file" for a complete discussion of index-only files.

Define Options (R I X) Specify options for AdmDefine when defining the file:

Blank	No change, DEFINE's default behavior
R	If the file exists, do a DEFINE /REDEF
I	Do a DEFINE /INIT
RI	Do a DEFINE /REDEF if the file exists, else do a DEFINE /INIT
X	If the file exists, delete the file before defining it
XI	Delete the file if it exists, then do a DEFINE /INIT

Selection Up to three lines of selection logic for the file (line 2 and line 3 are continuation lines). SELECT statements follow regular ADMINS Boolean logic syntax.

As in the attributes screens for other entity types, control information about entries and changes to file attributes are automatically maintained, and displayed at the bottom of the display screen.

From the file attributes screen, you may branch either to a screen used to specify the "file contains element" relationship, or to a screen for entering and updating text documenting the purpose of the file.

I.6.2 The File Contains Element Relationship

The "maintain file contains element" relationship screen is used to describe which data elements are contained in the file, and how the data elements are used (e.g. KEY/SORT order, derivation operators). In other words, this screen is where we give ADD the "DEF" for a file.

To enter, view, or update the relationship for a file, in the files overview screen place the cursor at the file you want, and then select the **DEF** option in the bar menu. You will branch to the "maintain file contains element" screen for the file.

I.6.2.1 File_contains_Element Relationship Screen

File attribute information for the file is displayed (File Type, # of records etc.). Up to 15 fields will be displayed at a time, in a way that looks identical to a traditional ADMINS "DEF" file. To add a data element (field) to the file relationship, type in the element (field) name, or use FIND to look up data elements (field names) defined in the Dictionary, and SELECT the ones you want. Once a data element name is entered or selected, its data format will be displayed.

You may then describe how the field is used in the file. Fill in the key/sort (KEY/SOR) order; if you do not want the whole field to be used for sorting, enter the number of significant bytes (SB). Fill in any aggregation operators where appropriate, and any secondary name used for the field.

Special keystrokes help make filling in this screen easier:

The **REMOVE** key deletes the field at the cursor, while the **INSERT** key opens a line to insert a field name. **NEXT_SCREEN displays the next page of fields in the file,** while **PREV_SCREEN** displays the previous page of fields.

I.6.2.2 File Contains Element Screen: The Menu Bar

The menu bar of the File Contains Element Screen contains the following options:

FLD_NO

Goto Field Number (Prompts for field number). If you type a higher field number than the current number of fields in the file, you are positioned at the next available field number.

COPY

Copy another file relationship Copy existing relationship to an empty one. This option is ignored if the DEF is not completely empty.

REMOVE

Removes elements from file relationship. Used to remove multiple or all elements from the file relationship. REMOVE prompts for the range of element numbers to be eliminated:

```
Begin field remove at Field number
(N to cancel)[1]:
```

Enter the field number of the beginning field in the range of fields to be removed. If nothing but RETURN is entered, the first field in the file will be the beginning of the range. To cancel the remove operation, enter "N". After the beginning field number is entered, ADD prompts for the last field number to be removed:

```
Stop field remove after Field
number (N to cancel)[25]:
```

Enter the field number of the last field in the range to be removed. If nothing but RETURN is entered, the last field in the file will be the end of the range. To cancel the remove operation, enter "N".

COMMIT

Commit Changes to Disk Write changes to disk. Changes are not saved (written to disk) until you COMMIT. If you attempt to exit the screen without saving changes, or you attempt to DEFINE a file without having saved the changes ADD will ask for confirmation that you wish to discard your changes.

ADD sends an informational prompt:

```
File DEF has been committed to
disk. Press Return to continue.
```

to inform you when your changes have been successfully written to disk.

DEFINE

Define Current File Definition. Create an ADMINS data file, and convert data if necessary. The current "committed" version of the DEF is used to create an ADMINS data file. If changes may have been made but not committed to disk, ADD will ask for confirmation before proceeding to define the file ignoring any uncommitted changes:

```
Changes not committed will not be
defined. OK to continue? (Y/N)[N]
```

If a file exists with the same file specification, ADD will MOVE/CONVERT the contents of the old file into the newly defined file.

The following options branch out of the File Contains Element Screen. If you exit the File Contains Element Screen without COMMITting your changes to disk, they will be lost. If changes may have been made but not committed to disk, ADD will ask for confirmation before branching.

```
Changes not committed will be lost. OK to continue? (Y/N)[N]
```

FIL_ATR	**File Attribute Screen**
OVRVIEW	**Overview of Files**
MENU	**Return to the main menu**

I.6.3 Descriptive Text for Application Documentation

Use the data file descriptive text screen to enter and view documenting information for this file. To enter the screen use the **DESCR.** option on the menu bar of the file attributes screen Up to 40 lines of up to 60 characters each may be entered to describe the purpose of this file. This screen functions in an identical manner to the data elements descriptive text screen described in Appendix I.4.5 “Descriptive Text for Application Documentation”.

I.6.4 DEMO: File attributes

In the preceding sections of the demonstration application we have described several data elements. We will now describe the attributes of a file, the customer file for the NTB order entry system, and then describe the **file contains date element relationship** for that file.

From the ADD main menu, enter "FA" to branch to the File Attribute screen.

The file attribute record entry for the customer file, CUSTOMER.MAS, is displayed below:

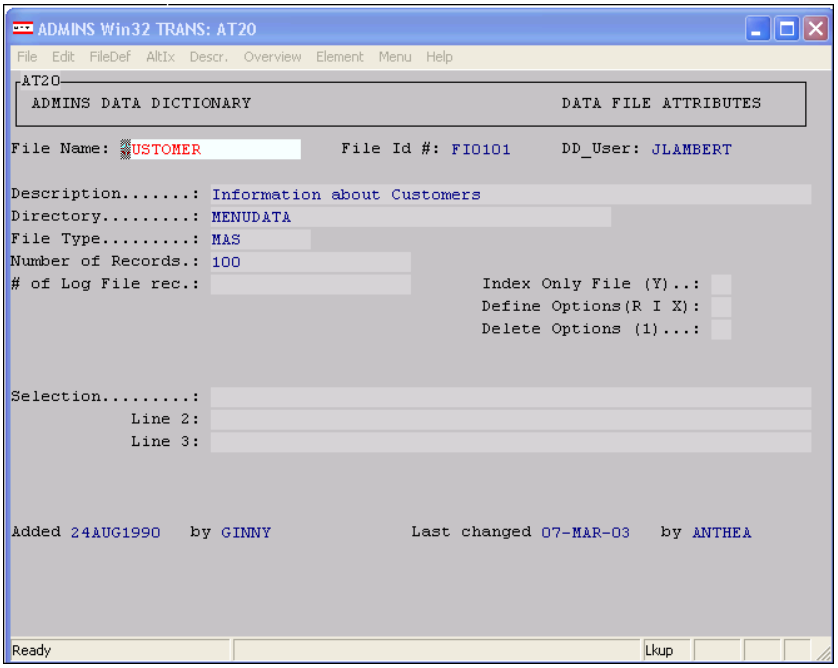


Figure I4-1 The CUSTOMER file attributes

I.6.5 DEMO: File contains data element

From the file attributes screen, select **DEF** on the menu bar to get into the **maintain file contains element relationship** screen.

The field names entered here must already have been identified as data elements. The following figure illustrates the use of Lookup to select a field.

```
FLD_NO  COPY    COMMIT  DEFINE  FIL_ATR OVRVIEW HELP    MENU
*RE02-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY       MAINTAIN FILE_contains_ELEMENT RELATIONSHIP |
*-------------------------------------------------------------------------*
File Name: CUSTOMER                Information about Customers
File Type: MAS        #Records: 100                        File ID: FI0101

 SEQ FIELD_NAME         FORMAT             KEY/SOR SB OPER    SECONDARY_NAME
  *-------------------------------------------------------------------------*
  |FIELD_NAME________ DATA_TYPE__________DESCRIPTION_____________________ |
  |7QTY               D                  Quantity Ordered                 |
  |CUSTID             XA9999             Customer Identification Code     |
  |CUSTOMER           A60                Customer Name                    |
  |DADDR1             A60                Delivery contact address line    |
  |DADDR2             A60                Delivery contact address line    |
  |DCITY              A40                Delivery contact address (City)  |
  |DCONTACT           A60                Delivery contact name            |
  |DELIVNOT           TI60               Delivery Notes                   |
  |DPHONE             A14                Delivery contact telephone number|
  |ORDER#             X999999            Order Number                     |
  *-------------------------------------------------------------------------*
  14
  15
```

Figure I4-2 Using Lookup to select fields.

The completed relationship appears below. Before we leave this screen we must be sure to **commit** our changes or else they will be lost.

```
FLD_NO  COPY    COMMIT  DEFINE  FIL_ATR OVRVIEW HELP    MENU
*RE02-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY       MAINTAIN FILE_contains_ELEMENT RELATIONSHIP |
*-------------------------------------------------------------------------*
File Name: CUSTOMER                Information about Customers
File Type: MAS        #Records: 100                        File ID: FI0101

 SEQ FIELD_NAME         FORMAT             KEY/SOR SB OPER    SECONDARY_NAME
   1 CUSTID             XA9999             KEY1
   2 CUSTOMER           A60
   3 DCONTACT           A60
   4 DADDR1             A60
   5 DADDR2             A60
   6 DCITY              A40
   7 DPHONE             A14
   8 SCONTACT           A60
   9 SADDR1             A60
  10 SADDR2             A60
  11 SCITY              A40
  12 SPHONE             A14
  13 DELIVNOT           TI60
  14
  15
```

Figure I4-3 The CUSTOMER file relationship.

I.7 Codelists

In the ADMINS Data Dictionary, **a codelist is a collection of code tables used for validation,** e.g. all valid department codes, object of expenditure codes, product codes, etc. The ADD facility includes an **"internal" codelist** (ADM_DD_CLIST.ADD) that is used to validate entries in the ADD screens themselves, and can also be used to store simple code tables for your applications. This internal codelist can store up to 9,999 different codelist tables. **"External" codelists** are regular ADMINS data files that are used as "repositories" for code tables. External codelists are useful for more complex or larger codelists, and especially when the information in a file used for other application purposes can also be used to validate new data as it is being entered. Unlike the internal codelist, external codelist repositories may contain only one codelist table.

ADD provides screens for describing and maintaining the individual tables in the internal codelist, and for identifying and describing external codelist repositories, the files they are stored in, and how they are to be used as tables.

ADD internal codelist table maintenance begins with the codelist overview screen, which displays a list of the codelist tables that have been identified. Place the cursor at the codelist table you wish to view or update, and then branch (via menu bar option **ATTRIB)** to the **codelist attributes detail screen.**

If you want to enter a new codelist table, branch to the codelist table attributes detail screen for any existing codelist table, then enter the name of the new codelist table (you'll be prompted "Enter "I" to INSERT".

I.7.1 Codelist Table Attributes Screen

For each codelist table the following attributes are displayed

Codelist Table Name	May be up to 24 characters, and must be unique within the current dictionary.
Description	Up to 60 characters that describe the purpose of the codelist table.
Codelist Repository	Data Dictionary ID of the codelist repository that contains this table. If an external repository is named, it must have already been described in the Codelist Repository attributes screen (see Appendix I.7.1.1 "Describing Codelist Repositories"). Press FIND to display a Lookup window that shows all the repositories (the internal codelist repository and all the external repositories you have entered) that are currently available in the data dictionary.
Codelist Data Format	The data format of the code field (maximum size of code stored in internal codelist is 24 characters). If you specify that the table is an stored in an external repository this field will automatically be loaded with the field type of the key field of the external repository file.

The remainder of the screen is used only when the table being described is stored in an external repository.

Codelist Key Field	The key field in the external file is also the "code" i.e. the key to the codelist table. This field is filled in automatically when you name the external repository file for this table (above).
Description Field	Enter the field in the external file that is to be used as the codelist table description.
UAC Field	Enter the field in the external file that is to be used as the codelist table user action field (see Appendix I.7.2.1 "Update Internal Codelist Tables ")

As in the attributes screens for other entity types, control information about entries and changes to codelist attributes are automatically maintained, and displayed at the bottom of the display screen.

From the codelist attributes screen you may branch either to a screen that presents an overview of the values for the codelist table being displayed (internal codelists only, see Appendix I.7.2 "Overview of Internal Codelist Table Values "), or to a screen for specification of the automatic lookup window for fields that validated against the codelist table (see Appendix I.7.3 "Automatic Lookup Windows ").

The menu bar option **DESCR branches to the Codelist Descriptive Text** screen, which is used to enter and update text documenting the purpose of the codelist.

I.7.1.1 Describing Codelist Repositories

Codelist Repositories are stored in regular ADMINS data files. Before you can describe how a codelist table is to be used as a codelist table, i.e. which fields are to be used the description field or the User Action Code field, you must first specify the attributes of the repository, including identifying the file that is going to be used to store it.

The **Codelist Repository Overview** screen displays a list of the Codelists Repositories that have been identified. Place the cursor at the repository you wish to view or update, and then branch (via menu bar option **ATTRIB)** to the **Codelist Repository Attributes screen.**

If you want to enter a new Codelist Repository, enter the name of the new Repository Codelist (you'll be prompted "Enter "I" to INSERT".

The Codelist Repository Attributes screen is used to identify and describe the ADMINS data file that is to be the basis for a codelist table. For each codelist the following attributes are displayed and/or entered:

Codelist Name	May be up to 24 characters, and must be unique within the current dictionary. This field will appear throughout ADD (i.e. in Lookup windows, etc.) whenever this Codelist Repository is referenced. **You should not enter the name of the data file that is to store the repository here.**
Description	Up to 60 characters that describe the purpose of this repository.

File Name	Any keystroke while the cursor is at this field "pops" and automatic Lookup window that displays all the files current specified in ADD. Select the file you want to be used as a codelist. **In order to be used as an Codelist Repository by ADD, the file's DEF must be specified using ADD, and it may have only one key field.**

The menu bar option **DESCR branches to the Codelist Repository Descriptive Text** screen, which is used to enter and update text documenting the purpose of the Codelist Repository.

Once the Codelist Repositories attributes have been described, you may specify how it is to be used in the Codelist Table Attributes screen, as described above in Appendix I.7.1 "Codelist Table Attributes Screen".

I.7.2 Overview of Internal Codelist Table Values

The overview of codelist table values screen displays the entries for the tables that are included in the internal codelist. To create or remove entries, or update existing entries use the menu bar option **UPDATE,** which branches to the update codelist tables screen.

I.7.2.1 Update Internal Codelist Tables

Use the Update Codelist Table screen to enter and change the entries in internal codelist tables. The **update codelist table** screen displays the DD_ID#, the codelist name, and the codelist description and has three editable fields:

Code	The code may be any data type, as defiend in the attributes screen for the codelist (but it is stored in the codelist file in character format).
Description	Up to 60 characters may be entered.A grid is provided in case you want to divide the description into subfields.
Action Code	Up to 16 characters may be entered. The user action code gives the application developer a way to store additional information to be associated with the code, which can be made available at "run time".

The **Description** and user **Action code** fields are automatically available in several ADMINS application situations, using a special D%fieldname or U%fieldname syntax, which means "get the description (D%) or the user action code (U%) for the value contained in fieldname from the codelist table associated with fieldname in the data dictionary". For example, if entries into a TRANS screen's field **DEPT** are automatically validated against the **department codelist table (i.e. if the table has been** associated with the DEPT field in the Data Dictionary), then if the field **D%DEPT** is placed in the screen's layout it is automatically loaded with the contents of the **description field** from the department table entry for the code entered in DEPT. Similarly, in REPORT, if **U%DEPT** is included in a layout section it will be loaded automatically with the contents of the **user action code field** from the department table entry for that record's DEPT code.

The D%/U% syntax is also supported in DISPLAY and SELECT sub-statements of TRANS LOOKUP paragraphs (see Section 5.11 "LOOKUP Window"), and in the /FIELDS qualifier of the IE/CREATE command (see Section 17.5 "IE: the ADMINS Import/Export Facility").

The two fields from the codelist table are also available to other ADMINS commands via the DDATTR subroutine (see Appendix H.10.1 "DDATTR: Get Data Dictionary Attributes & Codelists").

I.7.3 Automatic Lookup Windows

Use the **Codelist Table Lookup** screen to **create automatically-generated lookup windows** for any field that is validated against the codelist table. This screen displays the table name, the DD_ID#, the description for the codelist table, the names of the display fields (CODE, DESCRIPTION, and User Action Code for "internal" tables), and the data format of CODE.

To specify the automatic lookup window, enter a display width for each of the three fields (a zero display width will eliminate the field from the window), and enter the Title, Heading, and Footing you want for the window. ADD simulates a typical lookup window line at the bottom of the screen, using a series of C's to indicate the display width for code, D's for description, and U's for the action code.

You may also specify Lookup options **BOUND and** Key Search **CAPS.** By default both options fields are set to "N", i.e. neither option is in effect. These options are enabled if you enter "Y" in the option field.

Control information about entries and changes to lookup window specifications are automatically maintained, and displayed at the bottom of the display screen.

I.7.4 DEMO: Codelists

One of the relationships we must describe for the NTB order entry system is the file that corresponds to the order form itself. The ORDER file will include fields such as: ORDER#, CUSTID etc., plus several groups of fields to track the QUANTITY ordered of each ITEM in each PACKAGING style. These groups of fields, 1QTY 1ITEM 1PKG, 2QTY 2ITEM 2PKG, etc. will be based on prototype elements, QUANTITY, ITEM, and PACKAGING.

The prototype elements ITEM and PACKAGING are of particular interest because they (and consequently all the fields based on them) are to be validated against **codelists.**

Validation against a codelist is specified in the prototype element attributes screen when we are describing the prototype elements ITEM and PACKAGING, but before we can do that, we must first describe the attributes and contents of the codelists we will be referencing.

From the ADD main menu, select **CODELIST** on the menu bar, then select **ATTRIB** on the menu bar of the overview screen, to get into the codelist attributes screen.

New Tradition Bottling markets several flavors, and packages its products in several ways, as indicated in the following table:

Flavors	Packaging: 12 oz.	16 oz.	1 lit	2 lit	Can 12	6/ 12	6/ 16	6/ Can
Cola								
Diet Cola								
Cola Free								
Diet Cola Free								
Root Beer			XX	XX				
Diet Root Beer			XX	XX				
Club Soda		XX			XX		XX	XX

XX - Flavor not offered in this package style.

Two codelist tables, for Flavor and Packaging, are created in the data dictionary, as follows:

```
T_VALUES LOOKUP   DESCR    OVERVIEW HELP     MENU
*CL50-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY   User: GINNY                CODELIST ATTRIBUTES |
*-------------------------------------------------------------------------*
Codelist Table Name: FLAVOR                    DD_ID#: CT0101    INTERNAL

Description........: Product Identification List
Codelist Repository: CL0001  Data Dictionary Internal Codelist
Code Data Format...: X99

                     EXTERNAL CODELISTS ONLY
Codelist Key Field.:
Description Field..:
UAC Field..........:




Added 14-AUG-90   by GINNY           Last changed 14-AUG-90   by GINNY
```

Figure I5-1 Codelist attributes for FLAVOR

```
T_VALUES LOOKUP   DESCR    OVERVIEW HELP     MENU
*CL50-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY   User: GINNY               CODELIST ATTRIBUTES |
*-------------------------------------------------------------------------------*
Codelist Table Name: PACKAGING                 DD_ID#: CT0102    INTERNAL

Description........: Packaging Styles
Codelist Repository: CL0001  Data Dictionary Internal Codelist
Code Data Format...: X999

                     EXTERNAL CODELISTS ONLY
Codelist Key Field.:
Description Field..:
UAC Field..........:




Added 14-AUG-90   by GINNY           Last changed 14-AUG-90   by GINNY
```

Figure I5-2 Codelist attributes for PACKAGING

In the codelist attributes screen select **T_VALUES** in the menu bar to branch to the update codelist table screen. The codelists are built in this screen, entry by entry. Figure 5-3 below shows an entry for the FLAVOR table.

```
OVERVIEW ATTRIB   TABLES   HELP     MENU
*CL12-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        UPDATE CODELIST TABLES  |
*-------------------------------------------------------------------------------*

  Codelist: CT0101 FLAVOR
                  Product Identification List

              *------------------------*
         Code:|13                      |
              *---------1---------2----*    3         4         5         6
               1234567890123456789012345678901234567890123456789012345678901234567890
              *-------------------------------------------------------------*
  Description:|Diet Cola Free                                                |
              *----------------*-----------------------------------------*
  Action Code:|                |
              *----------------*
```

Figure I5-3 Entering values for Codelist FLAVOR

In some of the codelist entries for the PACKAGING table we utilize the user action code field to indicate which flavor/package combinations are **not** valid (see the Flavors/Packaging matrix above).

Figure I5-4 below shows the entry for PACKAGING code 113 (1 liter bottle). The Action Code field contains "20 21", the codes for the flavors (root beer and diet root beer) that are **not** available in 1 liter bottles.

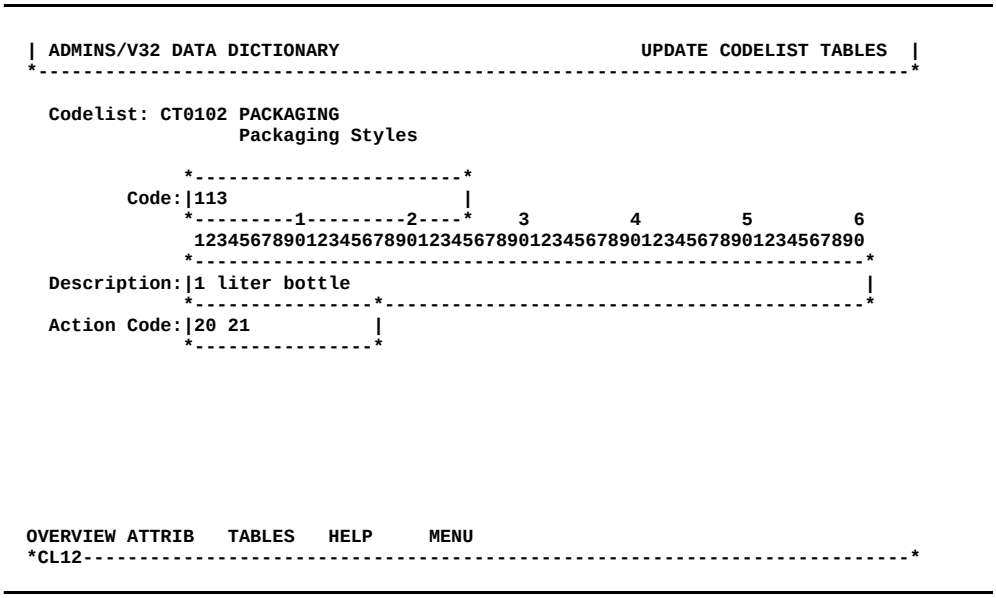


Figure I5-4 Entering a User Action Code

The user action codes entered will be automatically available for any field that is validated against the PACKAGING codelist table.

When the table entries are finished, select **OVERVIEW** in the menu bar to view the completed codelists. Figures I5-5 and PI5-6 show the completed codelists for FLAVOR and PACKAGING.

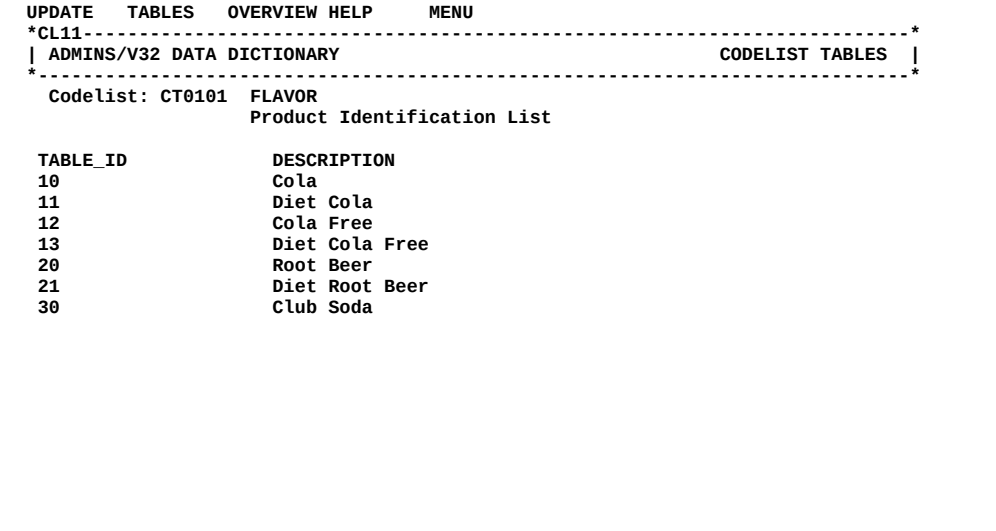


Figure I5-5 Codelist FLAVOR

```
UPDATE   TABLES   OVERVIEW HELP     MENU
*CL11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                                  CODELIST TABLES  |
*-------------------------------------------------------------------------------*
  Codelist: CT0102  PACKAGING
                    Packaging Styles

 TABLE_ID             DESCRIPTION
 111                  12 oz. bottle (individual)
 112                  16 oz. bottle (individual)
 113                  1 liter bottle
 114                  2 liter bottle
 121                  12 oz. can (individual)
 611                  12 oz. bottle (6-pack)
 612                  16 oz. bottle (6-pack)
 621                  12 oz. can (6-pack)
```

Figure I5-6 Codelist PACKAGING

I.7.5 DEMO: Automatic Lookup Windows

To specify automatic lookup windows for fields that are validated against the PACKAGING and FLAVOR codelists, select **LOOKUP** in the menu bar of the codelist attributes screen. Automatic Lookup windows for PACKAGING and FLAVOR are specified in Figures P5-7 and P5-8.

```
ATTRIB   T_VALUES OVERVIEW HELP     MENU
*CL52-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                          CODELIST TABLE LOOKUP|
*-------------------------------------------------------------------------------*
Codelist Table Name: FLAVOR                    DD_ID#: CT0101
Description........: Product Identification List

Display Fields..: CODE  X99                   DESCRIPTION   User Action Code
Display width...: 3                            20
Title...........:
Lookup Heading..:
 ID#      Flavor
Simulated Display
 CCC DDDDDDDDDDDDDDDDDDD
Footing.........:

Lookup Options
 Bound.........: N     (Y/N [N])
 Key Search CAPS: N     (Y/N [N])
Added 24-AUG-90   by GINNY            Last changed 24-AUG-90   by GINNY
```

Figure I5-7 Lookup for FLAVOR

```
ATTRIB   T_VALUES OVERVIEW HELP     MENU
*CL52-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                              CODELIST TABLE LOOKUP|
*-------------------------------------------------------------------------*
Codelist Table Name: PACKAGING                  DD_ID#: CT0102
Description........: Packaging Styles

Display Fields..: CODE  X999                  DESCRIPTION   User Action Code
Display width...: 4                            30
Title...........:
Lookup Heading..:
 Code       Packaging_Style
Simulated Display
 CCCC DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Footing.........:

Lookup Options
 Bound..........: N     (Y/N [N])
 Key Search CAPS: N     (Y/N [N])
Added 24-AUG-90   by GINNY            Last changed 24-AUG-90   by GINNY
```

Figure I5-8 Lookup for PACKAGING

I.7.6 DEMO: Codelist Repositories

In the DEMO portion of Appendix I.6 “Files” we showed how the Customer file relationship is specified. This file will be used to maintain information about NTB's customers, and as such, will be the main file in the NTB Customer screen, where records for new customers are entered, and the records for existing customer are kept up to date.

But this file can also be used as a codelist table for validation and lookup, because it contains records for all NTB's customers and has a single key field, CUSTID. This will have great utility in developing and using any screen in the NTB system that requires entry of the Customer ID field, such as the NTB Order Entry screen, as will be shown in the DEMO portion of Appendix I.10 “Application Development using ADD ”.

First we must specify that the Customer file is to be used as a Codelist Repository. From the Main Menu, branch to the **Codelist Repository Attribute** screen (Choice "CR"). The entry for the Customers Codelist Repository is created as follows (in the figure lookup is being used for entry into File Name):

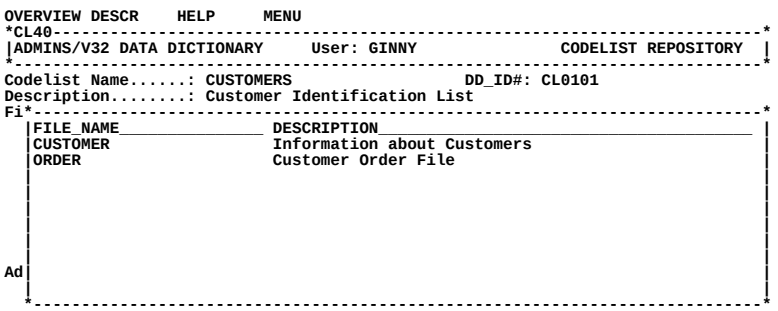


Figure I5-9 Specifying Codelist Repository Attributes

We then branch via the Main Menu (Choice "CA") to the **Codelist Table Attributes** screen, and enter the attributes of the Codelist Table that is based the "Customers" repository. Enter a description for the codelist table, and then enter the Codelist Repository name. Lookup is available:

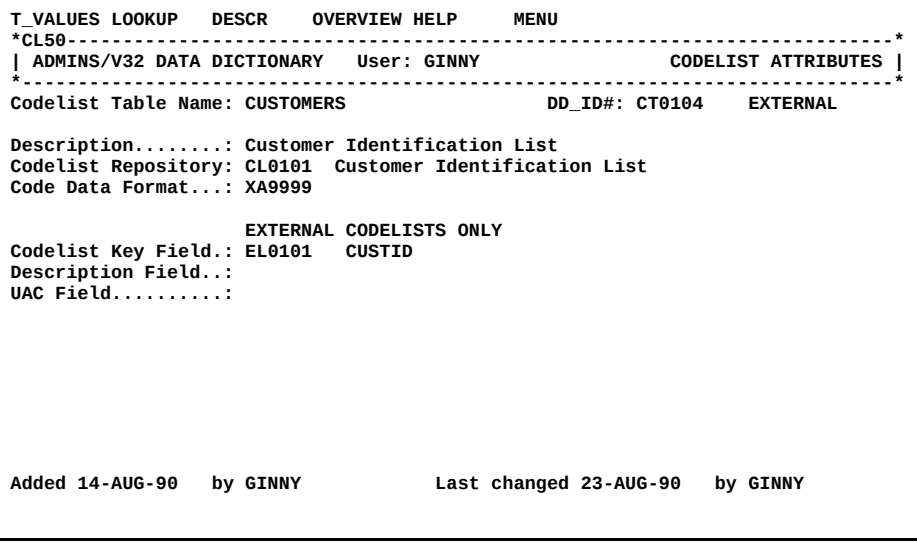


Figure I5-10 Codelist Tables Attributes - Specify Codelist Repository

When a Codelist Repository is entered, ADD will automatically fill in the Code Data Format (with the field type of the key field of the file that stores the Repository). The Codelist key field is also filled in automatically. To complete the description of how the data in the external file is to be used in this codelist table, specify which fields in the repository file are to be used as the Description Field and the UAC field (as described above for internal codelists, these fields will be available throughout the application for any field validated against this codelist table):

```
T_VALUES LOOKUP   DESCR    OVERVIEW HELP     MENU
*CL50-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY   User: GINNY               CODELIST ATTRIBUTES |
*-------------------------------------------------------------------------*
Codelist Table Name: CUSTOMERS                DD_ID#: CT0104    EXTERNAL

Description........: Customer Identification List
Codelist Repository: CL0101  Customer Identification List
Code Data Format...: XA9999

                     EXTERNAL CODELISTS ONLY
Codelist Key Field.: EL0101   CUSTID
Description Field..: EL0102   CUSTOMER
UAC Field..........: EL0108   DPHONE




Added 14-AUG-90   by GINNY           Last changed 23-AUG-90   by GINNY
```

Figure I5-11 Codelist Repository - The Customers Table

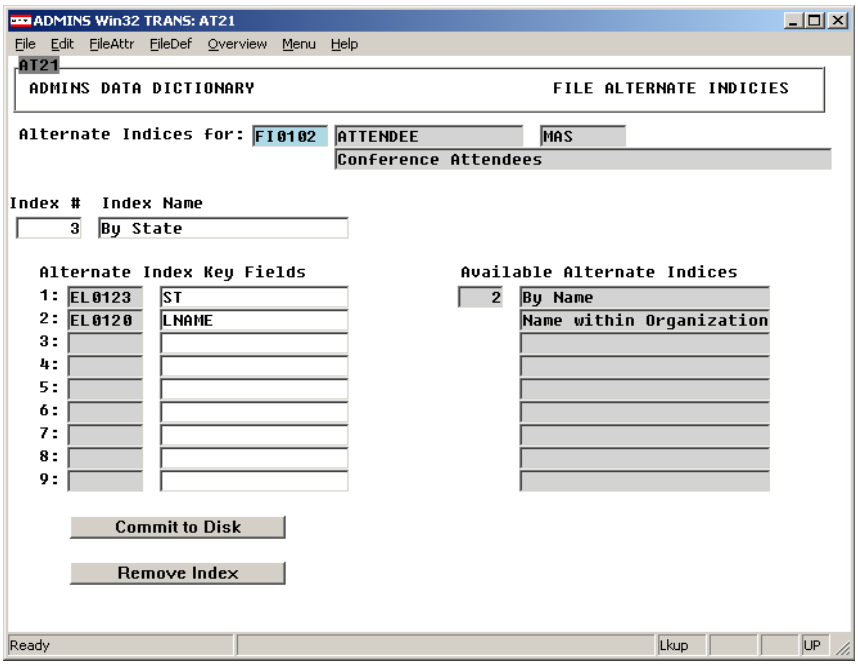
Having completed our specification of the Customers Table, we can go ahead and set up a lookup window for the table by branching to the **Lookup Specification** screen. The lookup specification is completed as follows:

```
ATTRIB   T_VALUES OVERVIEW HELP     MENU
*CL52-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                          CODELIST TABLE LOOKUP|
*-------------------------------------------------------------------------*
Codelist Table Name: CUSTOMERS                DD_ID#: CT0104
Description........: Customer Identification list

Display Fields..: CODE  XA9999               DESCRIPTION   User Action Code
Display width...: 5                           30
Title...........:
Lookup Heading..:
 Cust.      Customer_Name
Simulated Display
 CCCCC DDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
Footing.........:

Lookup Options
 Bound..........: N     (Y/N [N])
 Key Search CAPS: N     (Y/N [N])
Added 24-AUG-90   by GINNY           Last changed 24-AUG-90   by GINNY
```

Figure I5-12 Lookup specification for CUSTOMERS table.

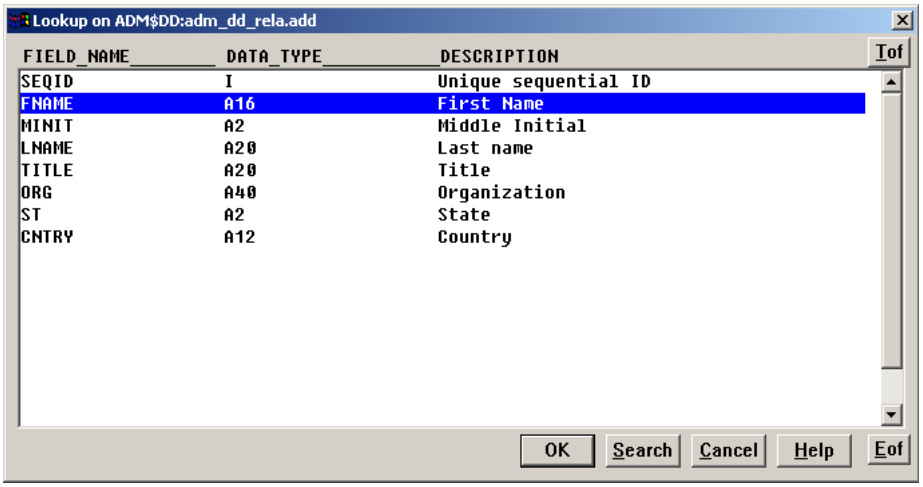
I.8 File Alternate Indices screen

By choosing the AltIx menu option from the **AT20 DATA FILE ATTRIBUTE** screen you can access the **AT21 FILE ALTERNATE INDICES** screen, which allows you to maintain Alternate Indices information for the file.



On the right hand side, this screen shows the number of alternate indices currently defined for this file, and their names.

On the left hand side, specify an index number. If it is an existing alternate index its name and key fields appear. If it is a non existing index you have the option of specifying its name and key fields. To specify key fields, use the lookup function to view and select fields from the file:



When all key fields are entered, click on the **Commit to Disk** button to save the alternate index in the dictionary.

If you need to remove a field from an alternate index, put the cursor in the field name field and press the **Delete** key.

I.8.1 Implementation

TRANS and ADED automatically maintain all active indices as records are added, deleted, or changed. For a 'batch' type command that changes or adds a significant number of records in a moderately sized file, it is more efficient to invalidate the indices, do its update, and then regenerate the indices using **SORT**.

I.8.2 (Use of Multi-Indexed Files

Multi-indexed files make it possible to access all the records in the file using any of the indices with the same efficiency as using the primary key. It will also allow e.g. REPORT to present the same data ordered in different ways without sorting the data.

One way to tell an ADMINS command to use one of the alternate indices to access a file is to put the index number (1-9) in the file option (e.g. N.MAS-M2 means open N.MAS in multi-user mode, and use index 2). A way to program around the TOTAL on KEY break in REPORT is to use syntax like:

```
FILE N.MAS-<Index #>
…
@@TOTAL1<Index #>.IND
…
@@TOTALn<Index #>.IND
```

In TRANS a user can switch to any index active for the file, unless the necessary fields are not present in the screen, or the developer places restrictions on the use of certain indices. When a user in TRANS (or ADED) asks to see the available indices, the index number and the index named specified in the DEF are shown.

I.9 Data Views

Data Views link data elements from one or more files together for easy and secure access. From a traditional ADMINS perspective, think of a Data View as a **pre-packaged set of virtual records that can be accessed throughout ADMINS as if it was a single, read-only ADMINS data file.**[4]

4. Data views are ALWAYS read-only, records cannot be appended, deleted, inserted, or updated. Data views cannot contain text fields, if text fields are contained in the files that make up the view, they are skipped when the view is built. (see Appendix I.9.2.3 "The View Relationship: FIELDS Screen ").

As with all ADD screen families, the **Overview** screen presents a list of all the views currently defined. For Data Views, you can branch directly from the Overview screen to the **Attributes** screen (menu bar option ATTRIB), or to the **View_contains_File/ Element relationship screens** (menu bar option DEF).

The **Attributes** screen is used to enter Data View attributes, and provides menu bar options to specify the view relationship, and to enter descriptive documentation for the view.

The **Data Views Descriptive Text** screen is used to enter and update text documenting the purpose of the data view.

View_contains_File/Element relationships are maintained using two related screens, the "FILES" screen (used to specify which physical files are used in the view, and how they are linked together) and the "FIELDS" screen (used to specify the fields from each file that are to be included in the view).

I.9.1 Data View Attributes screen

A Data View entity must be created, in this Data View attributes screen, before the Data View relationship can be laid out. For each Data View the following attributes are displayed:

View Name	Up to 24 characters long. Must be unique within the current dictionary. View Name is used in place of a file name when ADMINS commands access a view.
Description	Up to 60 characters that describe the purpose of the Data View.

As in the attributes screens for other entity types, control information about entries and changes to file attributes are automatically maintained, and displayed at the bottom of the display screen.

From the file attributes screen, you may branch either to a screen used to specify the "View contains File/Element" relationship, or to the description screen.

I.9.2 The View Contains File/Element Relationship

Data Views are maintained using two interrelated screens called **"FILES"** and **"FIELDS".** The FILES screen is used to specify which physical files are accessed in the view, and how these files are linked together. The FIELDS screen is used to identify the particular fields in each file that are to be available in the view.

I.9.2.1 The View Relationship: FILES Screen

A Data View Relationship is made up of a "main" file (File #1) and may contain one to nine "link" files (Files #2 through #10). The key fields of the main file become the key fields for the view.

The procedure for specifying a view is as follows:

1. Enter the main file of the view (File #1). Use **FIND** to display a Lookup Window of all the files available in the Dictionary.[5]

   ADD will automatically load all the non-text fields in File #1 into the view. Then ADD prompts as follows:

   `Enter/Update Fields from this file? (Y/N) [N]:`
   to see if you want to remove any of the fields in File #1 from the view. If you reply "Y" ADD will branch to the FIELDS screen (described in the following section).

2. Enter the names of the files you want linked into the Data View (use **FIND** to see the available files). As each link file is named ADD checks whether you want to load **all** the fields in that link file into the Data View:

   `Add the fields from this file to the View? (Y/N) [N]:`
   If you reply "Y" ADD will load all the non-text fields in the link file into the view. Then ADD prompts as follows:

   `Enter/Update Fields from this file? (Y/N) [N]:`
   to see if you want remove any of the link fields from the view (if all the fields were just added), or add link fields to the view (if you chose not to include them all). If you reply "Y" ADD will branch to the FIELDS screen.

3. For each link file, **you must specify the fields** that are to be used to make the links. **The key fields specified must already be part of the view,** i.e. for the first link file (File #2), the key fields must be in found in the main file (File #1); for the second link file (File #3), the key fields must be found in either the main file (File #1) or the first link file (File #2); and so on.

   To enter the link keys, place the cursor at the file name that you want, and then press the **RIGHT** arrow key.

5. In order for a file to be used in a Data View **it must have been defined using the Data Dictionary.**

I.9.2.2 FILES Screen: The Menu Bar

The menu bar of the Maintain Data View Relationship screen contains the following options:

COMMIT	**Commit Changes to Disk** Write changes to disk. Changes are not saved (written to disk) until you COMMIT. If you attempt to exit the two data view relationship screens without saving changes ADD will ask for confirmation that you wish to discard your changes. ADD sends an informational prompt: `Data View has been committed to disk. Press RETURN to continue.` to inform you when your changes have been successfully written to disk.
FIELDS	**View Fields Screen** Branch to the FIELDS Screen (described below).

The following options branch out of the Data View Relationship screens. If you exit the Data View Relationship screens without COMMITting your changes to disk, they will be lost. If changes may have been made in either the FILES screen or the FIELDS screen, but not committed to disk, ADD will ask for confirmation before branching.

```
Changes not committed will be lost. OK to continue? (Y/N)[N]
```

VIEW	View Attribute Screen
OVERVIEW	Overview of Views
MENU	Return to the main menu

I.9.2.3 The View Relationship: FIELDS Screen

You may enter the FIELDS screen for maintaining the Data View relationship either via an automatic branch after the

```
Enter/Update Fields from this file? (Y/N) [N]:
```

prompt, or via the **FIELDS** option on the menu bar of the FILES screen, or possibly after attempting to **COMMIT** a Data View relationship that has an error in its specification.

Branches to the FIELDS screen by default display the fields from the main file of the View. However if the branch is made while the cursor is at the "file name" field or "key field name" field for one of the link files, the FIELDS screen will display the fields for that link file. Once in the FIELDS screen you can display the fields for another file within the same view by entering either its file number (within the view) or its file name.

The following table describes the single-keystroke functions that may be used at any of the field-name fields.

FIND	Display Lookup window of available fields.
DOWNARROW or ENTER	Go to next field.
UPARROW	Go to previous field.
INSERT HERE	Open up a line to insert a new field between two existing fields.
REMOVE	Remove (delete) a field from the view.
PREV SCREEN	View the previous 20 fields.
NEXT SCREEN	View the next 20 fields.

To **rename a field when a duplicate field name occurs,** type the new name directly over the old name. It is important to recognize the distinction between providing a local name for a duplicate field name, and substituting a one data element for another. **Typing a new name for an existing field does not change the DDID,** but just applies a "local" pseudonym for that data element within the view to distinguish it from the View's other occurrence of its actual name. (Whenever the cursor goes to a renamed field, an informational message appears at the bottom of the screen showing the local name and the actual Data Element name.)

To **substitute one field for another** in a view, **you must first remove the unwanted field,** (use **REMOVE),** then **open a space for a new entry** (use **INSERT HERE),** and then type in the name of the field you want to substitute (or select it via the Lookup Window).

I.9.3 DEMO: Create Data View

An obvious Data View relationship for the NTB order entry system would be one that combines information about a specific order with information about the customer who made the order. This combination will probably be useful for many applications, but we'll need it specifically for a report we'll have to develop to generate order confirmations.

First, in the **Data View Attributes** screen, the new Data View name, CONFIRM, is entered, and the new entity is inserted.

```
VIEW_DEF DESCR.   OVERVIEW HELP     MENU
*AT30-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                          DATA VIEW ATTRIBUTES    |
*-------------------------------------------------------------------------*
View Name: CONFIRM              View Id #: DV0101     DD_User: GINNY

Description.......: NTB Customer Confirmation Order Form
Authorization code: 100









Added 23-AUG-90   by GINNY              Last changed 23-AUG-90   by GINNY
```

Figure I6-1 Create Entity for Data View CONFIRM

Menu bar option **VIEW_DEF** is then selected, to branch to the Maintain Data View Relationship screens.

In the FILES screen, ORDER is declared to be the main file of Data View CONFIRM (ADD will automatically load all the fields in ORDER into the view). The CUSTOMER file is then named as the first linked file. In this case its easiest to tell ADD to go ahead and load all the fields in CUSTOMER also. Figure I6-2 shows ADD asking whether all the fields in the just-named link file should be loaded (we reply "Y")

```
COMMIT   FIELDS   VIEW     OVERVIEW HELP     MENU
*RE30-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                  MAINTAIN DATA VIEW RELATIONSHIP |
*-------------------------------------------------------------------------*
View Name: CONFIRM                NTB Customer Confirmation Order Form

 ------------------------Files Defined in the View---------------------------
 File #   DDID   FILE_NAME          DESCRIPTION
      1  FI0102  ORDER              Customer Order File
      2  FI0101  CUSTOMER           *-----------------------------------*
                                    |                                   |
                                    |Key# File# Field Name          DDID |
                                    |---- ----- ----------------- ------|
                                    | 1:                                |
                                    | 2:                                |
                                    | 3:                                |
                                    | 4:                                |
                                    | 5:                                |
                                    | 6:                                |
                                    | 7:                                |
                                    | 8:                                |
                                    | 9:                                |
               Add the Fields from this file to the View? (Y/N) [N]: Y
```

Figure I6-2 After naming CUSTOMER as a Link file.

After loading all the linked fields, ADD inquires whether we want to branch immediately to the FIELDS screen. We reply "N" as we want to specify the key fields for the link.

The CUSTID field in the ORDER file will identify the record in the CUSTOMER file to be included in the CONFIRM Data View.

```
COMMIT   FIELDS   VIEW     OVERVIEW HELP     MENU
*RE30-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                  MAINTAIN DATA VIEW RELATIONSHIP |
*-------------------------------------------------------------------------*
View Name: CONFIRM                NTB Customer Confirmation Order Form

 ------------------------Files Defined in the View--------------------------
 File #   DDID   FILE_NAME          DESCRIPTION
      1  FI0102  ORDER              Information about Customers
      2  FI0101  CUSTOMER           *-----------------------------------*
                                     |                                   |
                                     |Key# File# Field Name          DDID |
                                     |---- ----- ------------------ ------|
                                     | 1:     1  CUSTID            EL0101|
                                     | 2:                                |
                                     | 3:                                |
                                     | 4:                                |
                                     | 5:                                |
                                     | 6:                                |
                                     | 7:                                |
                                     | 8:                                |
                                     | 9:                                |
                                     *-----------------------------------*
```

Figure I6-3 Specifying the key field for the link.

The **FIELDS** Option on the menu bar is then selected. Because the cursor was at the key fields for File #2, we are branched to the FIELDS screen with the fields for File #2 (CUSTOMER) displayed.

```
FILES    HELP
*RE30-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                  MAINTAIN DATA VIEW RELATIONSHIP |
*-------------------------------------------------------------------------*
View Name: CONFIRM                NTB Customer Confirmation Order Form
  *-RE31-----------------------------------------------------------------*
  |File  2: CUSTOMER                Information about Customers         |
  |-------------------------Fields Linked from File-------------------------|
  | Field   Field                          Field   Field                  |
  | Seq No   ID    Field Name              Seq No   ID     Field Name     |
  | ------  ------ ------------------      ------  ------  ------------- |
  |      1  EL0102 CUSTOMER                    11  EL0113  SPHONE        |
  |      2  EL0104 DCONTACT                                               |
  |      3  EL0105 DADDR1                                                 |
  |      4  EL0106 DADDR2                                                 |
  |      5  EL0107 DCITY                                                  |
  |      6  EL0108 DPHONE                                                 |
  |      7  EL0109 SCONTACT                                               |
  |      8  EL0110 SADDR1                                                 |
  |      9  EL0111 SADDR2                                                 |
  |     10  EL0112 SCITY                                                  |
  |                                                                       |
  |                                                                       |
  *-----------------------------------------------------------------------*
```

Figure I6-4 The FIELDS display for File #2 (CUSTOMER)

We could now use this screen to add or remove fields from the view. Lets assume we are happy with the view as it is, so we branch back to the FILES screen to **COMMIT** the specification to disk. Figure I6-5 below shows the result.

```
COMMIT   FIELDS   VIEW     OVERVIEW HELP     MENU
*RE30-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                  MAINTAIN DATA VIEW RELATIONSHIP |
*-------------------------------------------------------------------------*
View Name: CONFIRM                 NTB Customer Confirmation Order Form

 ------------------------Files Defined in the View---------------------------
 File #   DDID   FILE_NAME          DESCRIPTION
      1  FI0102  ORDER
      2  FI0101  CUSTOMER           *-----------------------------------*
                                     |                                   |
                                     |Key# File# Field Name          DDID |
                                     |---- ----- ----------------- ------|
                                     | 1:                                |
                                     | 2:                                |
                                     | 3:                                |
                                     | 4:                                |
                                     | 5:                                |
                                     | 6:                                |
                                     | 7:                                |
                                     | 8:                                |
PRESS "\" TO CONTINUE                                                      |
E-0370: Duplicate field name in use                                     --*
```

Figure I6-5 COMMIT: Duplicate field name error

The key field of file #2 has the same name as one of the fields in file #1.

This kind of name conflict will occur quite often because the field being used to make the link in the "main" file is likely to be the same data element (the same DDID) as the key of the link file. In most of these cases the simplest solution is to remove the field from the linked fields list in the FIELDS screen (in traditional ADMINS terminology, there is ordinarily no reason to link in the key field of the link file).

For the more general case, where the naming conflict does not involve the key field of the link file, the application developer has two options:

1. Remove the field name from the list of fields for one of the files.
2. Rename the field in the list of fields for one of the files (by typing over its name, as explained in Appendix I.9.2.2 “FILES Screen: The Menu Bar ”).

Then return to the FILES screen and **COMMIT** the specification to disk.

Note that the text field DELIVNOT in the customer file is not loaded because text fields are not supported in data views.

I.10 Application Development using ADD

The preceding sections of this document explain how the developer describes the various components of an application to the ADMINS Data Dictionary. There has been little mention of how these components will be arranged into a production system. In the demonstration application we have yet to specify a single screen or report for New Tradition Bottling's Order Entry system.

This underscores one very significant characteristic of ADD-based application development: **out-front data base design in ADD-based applications is essential;** while in conventional ADMINS applications it is a matter of choice. Out-front, rigorous data base design is a sound technique, even in the conventional ADMINS environment. But the ADMINS tools also provide a rich environment for (and consequently encourage) quick and easy prototyping, "use-one-time" files, ad hoc screens, procedures, and reports, etc. While this may help overloaded development staff keep their users happy, or at least at bay, it can also result at worst in a documentation and maintenance disaster, and at best in duplication, inefficiency, and management headaches.

Using the ADMINS data dictionary is a choice in favor of documented, maintainable code, and more easily managed applications, while perhaps giving up some of the traditional ADMINS "quick-fix", ad hoc, and prototyping capability.

In what follows we will specify the first screens of the production system, and in doing so, begin to realize the benefits of the thorough data base design methodology that the ADMINS data dictionary requires.

I.10.1 DEMO: Wrapping up the Database Design

As was discussed in Appendix I.7.4 “DEMO: Codelists ”, one of the relationships we must describe for the NTB order entry system corresponds to the order form itself, and includes groups of fields (1ITEM 1PKG 1QTY, 2ITEM 2PKG 2QTY, etc.) that are based on the prototype elements, FLAVOR, PACKAGING, and QUANTITY. The prototype elements FLAVOR and PACKAGING, in turn, are associated with internal codelists of the same name. Appendix I.7 “Codelists” we have shown how these two code lists are populated.

To complete the data base design for the New Tradition Order Entry system we must perform the following:

1. Enter the prototype elements FLAVOR, PACKAGING, and QUANTITY, associating FLAVOR and PACKAGING with the internal codelists of the same name.

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT01-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                    PROTOTYPE ELEMENT ENTRY SCREEN |
*-------------------------------------------------------------------------*
Prototype Field Name: FLAVOR            DD_ID#: PE0105 DD_User: GINNY

Data Format.......: X99
Description.......: Product Identification
Def. display width:  2
Justification(L/R): L
Default Heading 1.: Item                        Line Label: Item
Default Heading 2.:

Codelist Table....: FLAVOR
     Error Message:
            CT0101  Product Identification List
Validation rules..:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):

Added 24-AUG-90   by GINNY             Last changed 24-AUG-90   by GINNY
```

Figure I7-1 Prototype Element FLAVOR

2. Enter seven groups of data elements, 1ITEM 1PKG 1QTY through 7ITEM 7PKG 7QTY based on the FLAVOR, PACKAGING and QUANTITY prototypes.

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------*
Field Name: 1ITEM              DD_User: GINNY            Data Element #: EL0116
                            Prototype: FLAVOR           Prototype DDID: PE0105
Data Format.......: X99
Description.......: Product Identification
Def. display width:  2
Justification(L/R): L
Default Heading 1.: Item                       Line Label: 1Item
Default Heading 2.:

Codelist Table....: FLAVOR
     Error Message:
             CT0101 Product Identification List
Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):

Added 24-AUG-90   by GINNY             Last changed 24-AUG-90   by GINNY
```

Figure I7-2 Data Element 1ITEM

3. Build an internal codelist table, SALESREP, with automatic lookup. Then enter a data element SALESREP to be validated against the SALESREP table.

```
UPDATE   TABLES   OVERVIEW HELP     MENU
*CL11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                                 CODELIST TABLES  |
*-------------------------------------------------------------------------*
  Codelist: CT0103  SALESREP
                    Sales Representatives

 TABLE_ID            DESCRIPTION
 01                  Albert Allard
 02                  Beatrice Beston
 03                  Cosmo Carducci
 04                  Douglas Dillon
 05                  Eduardo Estavez
 06                  Filipe Fetisov
 07                  George Goodman
 08                  Harry Ho
 09                  Ida Ianello
 10                  Jan Johnson
```

Figure I7-3 Codelist Table SALESREP

4. Build the ORDER file relationship.

```
FLD_NO  COPY    COMMIT  DEFINE  FIL_ATR OVRVIEW HELP    MENU
*RE02-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY       MAINTAIN FILE_contains_ELEMENT RELATIONSHIP |
*-------------------------------------------------------------------------*
File Name: ORDER                    Customer Order File
File Type: MAS        #Records: 100                          File ID: FI0102

 SEQ FIELD_NAME         FORMAT            KEY/SOR SB OPER    SECONDARY_NAME
   1 ORDER#             X999999           KEY1
   2 PAGE               I                 KEY2
   3 CUSTID             XA9999
   4 1ITEM              X99
   5 1QTY               D
   6 1PKG               X999
   7 2ITEM              X99
   8 2QTY               D
   9 2PKG               X999
  10 3ITEM              X99
  11 3QTY               D
  12 3PKG               X999
  13 4ITEM              X99
  14 4QTY               D
  15 4PKG               X999
```

Figure I7-4 The ORDER File

5. Define the ORDER and CUSTOMER files.

```
FLD_NO  COPY    COMMIT  DEFINE  FIL_ATR OVRVIEW HELP    MENU
*RE02-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY       MAINTAIN FILE_contains_ELEMENT RELATIONSHIP |
*-------------------------------------------------------------------------*
File Name: CUSTOMER               Information about Customers
File Type: MAS        #Records: 100                          File ID: FI0101

 SEQ FIELD_NAME         FORMAT             KEY/SOR SB OPER    SECONDARY_NAME
   1 CUSTID             XA9999             KEY1
   2 CUSTOMER           A60
   3 DCONTACT           A60
   4 DADDR1             A60
   5 DADDR2             A60
   6 DCITY              A40
   7 DPHONE             A14
   8 SCONTACT           A60
   9 SADDR1             A60
  10 SADDR2             A60
  11 SCITY              A40
  12 SPHONE             A14
  13 DELIVNOT           TI60
  14
  15
```

Figure 7-5 Defining the CUSTOMER file.

I.10.2 DEMO: Screen Specification

Figure I7-6 shows the order entry screen for New Tradition Bottling:

```
Query    Customer Exit

                    NEW TRADITION BOTTLING
                    O R D E R    E N T R Y
         Order: 000155  Customer ID: B0035          Sales Rep: 05
         Page: 1       Circle Variety Haverford     Eduardo Estavez
              *-------------------------------------------------*
              |    Item             Packaging     Quantity      |
              *-------------------------------------------------*
              | 1. 11               611                4      |
              |    Diet Cola        12 oz. bottle (6-pack)     |
              | 2. 30               121                2      |
              |    Club Soda        12 oz. can (individual)    |
              | 3. 21               621                2      |
              |    Diet Root Beer   12 oz. can (6-pack)        |
              | 4.                                             |
              |                                                |
              | 5.                                             |
              |                                                |
              | 6.                                             |
              |                                                |
              | 7.                                             |
              |                                                |
              *-------------------------------------------------*
```

Figure I7-6 The order entry screen.

Figure I7-7 illustrates automatic validation against a codelist in the order entry screen. Figure I7-8 illustrates use of the automatic lookup window feature.

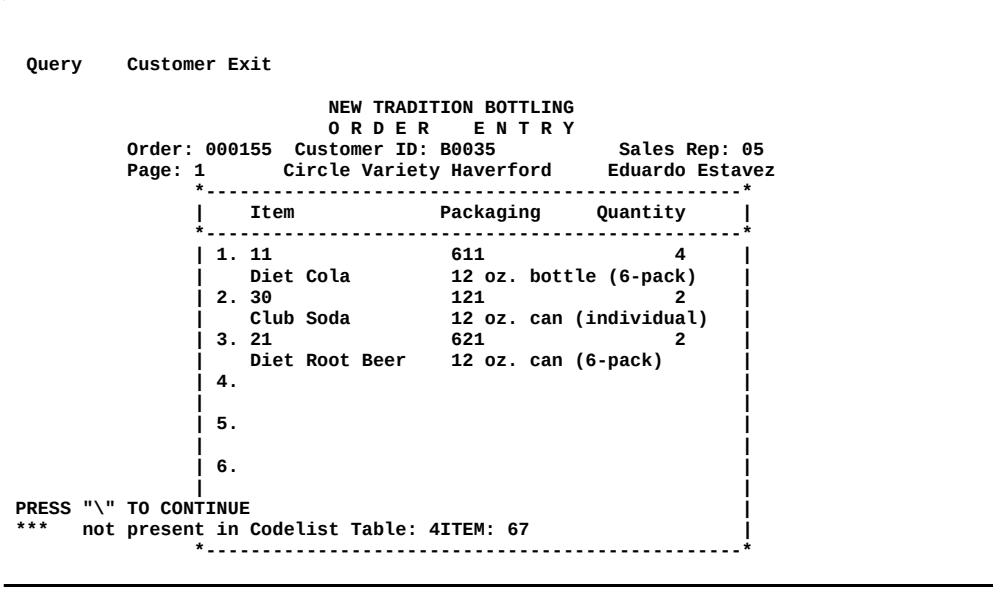


Figure I7-7 Validation against a Codelist Table

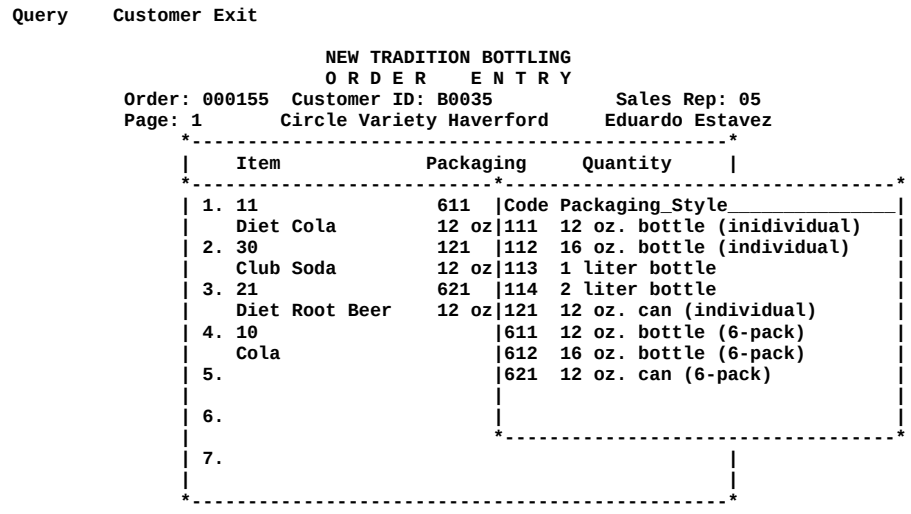


Figure I7-8 Automatic Lookup window

Note in the following listing of ORDER.TRS, the source code for the order entry screen, that no specification need be made either for check statements to validate entries against the codelists, or for the automatic lookup windows. (The check statements that do appear test that fields are entered in the right order, and that the item/packaging combination entered is valid. This testing for valid combinations is performed in ORDER.RMS utilizing the "U%fieldname" User Action Codes from the

codelist entries, and is discussed below). Also note that the "D%fieldname" syntax is used to display the description for a "fieldname" when a field is associated with a codelist table (i.e. D%SALESREP will contain the description for the code value entered into SALESREP).

```
*
* Order entry screen specification.
*
ORDER NTB_DATA:ORDER.MAS 1 ORDER.RMO INSERT DELETE NOMSG
*
* Display keys in reverse video,
* editable fields in bold
*
VIDEO KEYS REVERSE EDIT BOLD
*
* Screen Header
*
V HDR1/A23 %BOLD+REV 'NEW TRADITION BOTTLING'
V HDR2/A23 %BOLD     'O R D E R    E N T R Y'
*
* Keys are ORDER# and PAGE
* up to seven item/package combinations
* can be ordered on a page (mimics order forms
* filled out by sales rep in the field.)
*
E ORDER#
E PAGE
*
* Data values are customer id,
* and sales rep, then the order
* itself (Item/Package and Quantity)
*
E CUSTID
E SALESREP
*
E 1ITEM
E 1PKG
E 1QTY
*
E 2ITEM
E 2PKG
E 2QTY
*
E 3ITEM
E 3PKG
E 3QTY
*
E 4ITEM
E 4PKG
E 4QTY
*
E 5ITEM
E 5PKG
E 5QTY
*
E 6ITEM
E 6PKG
E 6QTY
*
E 7ITEM
E 7PKG
E 7QTY
*
DR ERR/I
*
C ERR EQ 100
This item is not available in the packaging style specified.
*
C ERR EQ 101
You must enter the item code before you enter the packaging code.
*
BOX DEFAULT
*
```

```
* Menu Bar spec
*
BAR 1 OPTIONS=VISIBLE
   Query BRANCH Q
   Customer Query
   Customer BRANCH C
   Customer Entry
   Exit QUIT
   Back to your menu
*
SCREEN
BL
BL
DW HDR1-----------------
DW HDR2-----------------
Order: ORD---  Customer ID: CUST--    Sales Rep: SA-
 Page: PAGE-   D%CUST---------------------- D%SAL----------------
               +================================================+
               !    Item          Packaging            Quantity !
               +================================================+
               ! 1. 1I-               1P-           ---1QTY     !
               !    D%1ITEM---------- D%1PKG------------------ !
               ! 2. 2I-               2P-           ---2QTY     !
               !    D%2ITEM---------- D%2PKG------------------ !
               ! 3. 3I-               3P-           ---3QTY     !
               !    D%3ITEM---------- D%3PKG------------------ !
               ! 4. 4I-               4P-           ---4QTY     !
               !    D%4ITEM---------- D%4PKG------------------ !
               ! 5. 5I-               5P-           ---5QTY     !
               !    D%5ITEM---------- D%5PKG------------------ !
               ! 6. 6I-               6P-           ---6QTY     !
               !    D%6ITEM---------- D%6PKG------------------ !
               ! 7. 7I-               7P-           ---7QTY     !
               !    D%7ITEM---------- D%7PKG------------------ !
               +================================================+
*
* (Branch paragraph follows - not included)
*
```

I.10.3 DEMO: ORDER.RMS

Figure I7-9 demonstrates the checking for valid item/packaging combinations that is done in ORDER.RMS, utilizing the user action code field from the PACKAGING codelist.

```
Query    Customer Exit

         NEW TRADITION BOTTLING
         O R D E R    E N T R Y
Order: 000261  Customer ID: A0034           Sales Rep: 03
 Page: 1       Highland Ave. Superette      Cosmo Carducci
               *-----------------------------------------------*
               |    Item             Packaging     Quantity    |
               *-----------------------------------------------*
               | 1. 10               111               6     |
               |    Cola             12 oz. bottle (inidividua |
               | 2. 20               121               5     |
               |    Root Beer        12 oz. can (individual)   |
               | 3. 30               121                       |
               |    Club Soda                                  |
               | 4.                                            |
               |                                               |
               | 5.                                            |
               |                                               |
               | 6.                                            |
               |                                               |
RESS "\" TO CONTINUE                                           |
his item is not available in the packaging style specified.    |
               *-----------------------------------------------*
```

Figure I7-9 Check Statement for Item/Packaging Combination.

Note in the following listing of ORDER.RMS, that the fields U%1PKG, U%2PKG etc. (the "user action codes" available from the codelist PACKAGING that has been associated with the fields 1PKG, 2PKG, etc.) are available automatically in the RMO. This RMO checks for valid item/packaging combinations by determining if the item code is found in the user action field of the packaging code, if it is the error condition is set.

```
FILE NTB_DATA:ORDER.MAS
*
S$S/A6
M$M/A2
*
B$B/A2
*
ERR/I
*
*  OGRP,OFLD used to break down field name into
*  components, i.e. for field "1PKG"
*  OGRP will be "1", OFLD will be "PKG".
*
OGRP/A1
OFLD/A10
*
* UPCT used to build action code field name,
* i.e. for field "1PKG" action code field name
* is "U%1PKG". UFLD holds resulting field name (for GETFLD)
* UACT holds value obtained via GETFLD.
*
UPCT/A2 'U%'
UFLD/A10
UACT/A16
*
* ITEMEND used to build field name of item field.
```

```
* CITEM holds resulting name for GETFLD.
* UITEM holds value from GETFLD.
* UIALP holds value converted to alpha.
* UCHK is where UIALP string is found in UACT
* (if its found ERR is set to 100)
*
ITEMEND/A4 'ITEM'
CITEM/A10
UITEM/X99
UIALP/A2
UCHK/I
GSTAT/I
GOFS/I
*
PROGRAM
*
* Don't do anything pre-link.
*
IF (M$M EQ 'UX' OR 'IX') THEN STOP ; END
*
* Go directly to update mode after "I to Insert"
*
IF (M$M EQ 'IN') AND (S$S EQ 'BEGREC') THEN B$B = 'LF' ;
 STOP ; END
*
* Process in update mode only
*
IF (M$M NE 'UP') THEN STOP ; END
*
* Initialize error flag.
* Use OFLD to check if a "Packaging" field has
* been entered (Stop processing otherwise). Load OGRP
* with which packaging field (number) has been entered.
*
ERR = 0
*
OFLD = STR(OFLD,S$S,2,4) ;
 IF OFLD NE 'PKG' THEN STOP ; END ;
 OGRP = STR(OGRP,S$S,1,1)
*
* Field name of item field corresponding
* to packaging field just entered is put into CITEM.
* If item field is empty exit with "Enter item first"
* message. GETFLD loads value of CITEM into UITEM,
* UIALP is UITEM converted to alpha.
*
CITEM = NCAT(CITEM,OGRP,ITEMEND) ;
 GSTAT = GETFLD(UITEM,CITEM,GOFS) ;
 IF UITEM EQ '00' THEN ERR = 101 ; STOP ELSE ERR = 0 END ;
 UIALP = NCAT(UIALP,UITEM)
*
* Field name of user action code
* for field just entered is put into UFLD,
* GETFLD loads value of UFLD into UACT
*
UFLD = NCAT(UFLD,UPCT,S$S) ;
 GSTAT = GETFLD(UACT,UFLD,GOFS) ;
*
* If item code is found in user action field
* of package code, then set error condition
* (Item not available in that package!).
*
UCHK = UACT INCL UIALP ;
 IF UCHK NE 0 THEN ERR = 100 ELSE ERR = 0 END
*
STOP
```

I.11 Data Dictionary Reports and Where Used Analysis

Two of the choices offered by the Data Dictionary main menu provide information about the entities and relationships contained in the Dictionary. Item **DR** "Data Dictionary Reports" allows you to select from a series of menus to produce brief or detailed "catalogues" of all the entries in your dictionary for a particular entity type, to produce listings of file and data view relationships and codelist table values, or to produce a listing of the relationships for each entity (a "Where Used" report). Item **WH** "Where Used Screen" displays all the relationships that each entity is involved in.

I.11.1 Data Dictionary Reports

If you select main menu item DR the following menu is displayed:

```
          Branch Menu
 1 : Prototype Element Reports
 2 : Data Element Reports
 3 : Data File Reports
 4 : Codelist and Codelist Table Reports
 5 : Data View Reports
  6 : Entity Where Used Analysis
```

Enter the number of the option you want to select, or using the up and down arrow keys move the cursor to the option you want, then select it by pressing DO. Each of these options leads to a screen menu:[6]

```
Prototype Element Reports

 1: Catalog of Prototype Elements
 2: Catalog of Prototype Elements, Full Listing

Data Element Reports

 1: Catalog of Data Elements
 2: Catalog of Data Elements, Full Listing

Data File Reports

 1: Catalog of Data Files
 2: Catalog of Data Files, Full Listing
 3: File Definition (.DEF)
 4: File Definition (.DEF) For All Files, Sorted by File Name
 5: File Definition, Sorted by Field Name
 6: File Definition, for DEFINE processing
```

6. File definitions may be produced from the Data File Report Menu, or by directly calling the file definition report, e.g. "$ REPORT ADM$DD_DIST:ADM_DD_RDEF/RPO" at the DCL prompt or in a command procedure. Before ADM_DD_RDEF.RPO is run directly, two logical names it uses must be assigned: ADM$DD_FILE identifies the Data Dictionary file relationship that you want to produce a DEF for, e.g. "ORDER" or "CUSTOMER". ADM$DD_FILEDEF specifies what name (e.g. "NEWORDER.DEF" or "X$DISK:[SALES]XCUST.DEF") will be given to the DEF instruction file produced by the report.

```
Codelist and Codelist Table Reports

1: Catalog of Codelist Tables
2: Catalog of Codelist Table Values
3: Catalog of Codelist Table Values, Full Listing

Data View Reports

1: Catalog of Data Views
2: Catalog of Data Views, Full Listing
3: Data View Definition

Entity Where Used Analysis

1: Entity Where Used Analysis
```

In any of these menus, select the report you want by entering its number at the "Your Choice" prompt. The report output is sent to ADM$SPOOL0. To exit back to the main menu, type M at the "Your Choice" prompt.

I.11.2 Where Used Screen

Main menu selection WH "Where Used Screen" provides the same information as the "Entity Where Used Analysis" report, but in an on-screen format that is more convenient for quickly checking the impact of a change to a single entity.

```
HELP     MENU
*RI02-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        ENTITY WHERE USED OVERVIEW |
*-------------------------------------------------------------------------*

Entity: EL0101 CUSTID                  Customer Identification Code

 DDID    SEQ   ENTITY_NAME             DESCRIPTION
DV0101  1,003  CONFIRM                  NTB Customer Confirmation Order Form
FI0101      1  CUSTOMER                 Information about Customers
FI0102      3  ORDER                    Customer Order File
```

Figure I8-1 Where Used Screen: Data Element CUSTID

To view the relationships that utilize a given entity, press FIND to display a Lookup window that shows all the entities currently in the data dictionary, then SELECT the entity you want.

I.12 Setup for the ADMINS Data Dictionary

This section describes the logical names, symbols, and data files used by the ADMINS Data Dictionary. It also describes how to use command procedures provided by ADMINS to help you set up the ADMINS Data Dictionary application.

I.12.1 Logical Names and Symbols

In order to run the Dictionary commands, the following logical names must be assigned:

ADM$DD_DIST	Points to the directory where the ADMINS Data Dictionary application itself (i.e. the .TROs, .RMOs, .HLP etc.) resides. New versions of this application will be released with new releases of ADMINS. The contents of this directory should not be modified.
ADM$DD	Points to the directory where the Data Dictionary data files (.ADD files) that describe your application are found. ADMINS supplies data files that are empty except for internal Data Dictionary entities. These "empty" .ADD files should not be altered; they should be kept as a "clean" template that can be copied to start new Data Dictionary applications. To create a new Dictionary, copy the clean template files to your working application directory, and then make sure ADM$DD points to that working application directory.
ADM$DD_LOAD	If you want to update a Data Dictionary by converting an existing application (See Appendix I.13 "Converting Applications to the ADD Environment "), this logical name must be assigned. ADM$DD_LOAD is used to identify the directory that will hold the temporary files used in the conversion procedures.

The following symbol must exist:

```
ADD :== 'TRA ADM$DD_DIST:ADM_DD_MENU
```

ADD opens the ADMINS Data Dictionary main menu in TRANS. All dictionary functionality can be reached from the main menu.

I.12.2 The ADMINS Data Dictionary Files

The ADMINS Data Dictionary automatically maintains six main files:

ADM_DD_NAME.ADD	The Name Index file. Allows access to all the Data Dictionary entities by entity type and name.
ADM_DD_ATTR.ADD	The Attribute file. Contains all the attributes of the entities defined in the Dictionary. The Attribute File is keyed by the Data Dictionary Id Code for the Entity.
ADM_DD_RELA.ADD	The Relation File. Describes all the relations defined in the Dictionary, e.g. "FILE_contains_ELEMENT", "VIEW_contains_FILE/ELEMENT".
ADM_DD_RELI.ADD	Inverted Relation file. Gives all the WHERE_USED relations.
ADM_DD_CLIST.ADD	Data Dictionary Codelist File. Contains all the internal codelist tables.
ADM_DD_CTRL.ADD	Data Dictionary Control file.

The major key field that ties all the files together is the DDID field, which has a format of XAA9999, where the 'AA' portion identifies the entity type, e.g. 'EL' for Data Element, or field, 'FI' for File, 'DV' for Data View, etc., and the '9999' portion is a sequence number within the Entity type. DDID numbers are assigned automatically when a new entity is entered into the Dictionary.

All entities within the same entity type must be assigned a unique name (e.g. a field and a file can have the same name, but every field must have a different name). This unique name, through the ADM_DD_NAME.ADD file, provides an access path to the Entity and all of its Relations. The ADM_DD_NAME.ADD file contains the DDID value for the Entity, which is the main key into the ADM_DD_ATTR.ADD and ADM_DD_RELA.ADD files. ADMINS data file headers will contain the DDID value for the fields, giving, for example, a screen (TRO) a direct access path into the ADM_DD_ATTR.ADD file to pick up any information required, e.g. validation rules, codelists, etc.; or giving reports access to default headings.

I.12.3 Setup Procedures for the ADMINS Data Dictionary

When the ADMINS release tape is loaded onto the system, you must specify the disk/directory where the Data Dictionary files are to be loaded.[7]

That directory contains the DCL command procedure **SETUP.COM,** which can be used to set up the ADMINS Data Dictionary environment. SETUP.COM accepts the location of the Data Dictionary .ADD files[8] and the Data Dictionary application as command line arguments, e.g.:

```
$ @setup DUA0:[ADD] DUA0:[DD_DIST]
```

would set DUA0:[ADD] as the location of the .ADD files, and DUA0:[DD_DIST] as the location of the Data Dictionary application. If the command line arguments are omitted SETUP.COM will prompt for them:

```
$ @setup
Where are the .ADD files? dua0:[add]
Where are the Data Dictionary TROs RMOs etc.? dua0:[dd_dist]
```

SETUP.COM uses the responses to make the ADM$DD_DIST and ADM$DD logical name assignments and then sets the ADD symbol:

```
ADD :== 'TRA ADM$DD_DIST:ADM_DD_MENU
```

I.12.3.1 Setting up the ADD DEMO Application

When the ADMINS release tape is loaded onto the system, you must specify the disk/directory where the Data Dictionary Demonstration Application files are to be loaded.[9]

To set up the Demonstration application set the default directory to the location of the DEMO files and run DEMOSETUP.COM, e.g.:

```
$ set default DUA0:[ADD_DEMO]     $ @DEMOSETUP
    ------------------------------------------------------

        This procedure calls the command procedure

                ADM$DD_DIST:SETUP.COM

        to define the environment for the

        ADMINS Data Dictionary Demonstration Application

        Answer the following prompts to specify:

        What disk/directory the ADMINS Data Dictionary
        (i.e. the Dictionary TROs and RMOs etc.)
        has been loaded into (the procedure will assign your
        response to the logical name ADM$DD_DIST).

        The procedure also assigns the current default disk
        and directory to the logical names NTB_DATA and ADM$DD
    ------------------------------------------------------

 Where is the ADMINS Data Dictionary:
```

7. See the **ADMINS Distribution Guide** that comes with your distribution kit.
8. Please note: the location for the .ADD files that you give to SETUP.COM should be the location where **copies** of the original "template" ADD files have been placed. **NOT** the location of the original template files.
9. See the **ADMINS Distribution Guide** that comes with your distribution kit.

Respond with the location of the Data Dictionary Application:

```
Where is the ADMINS Data Dictionary: DUA0:[DD_DIST]
```

If the adm$dd_dist logical name already assigned when you run DEMOSETUP, it prompts as follows:

```
adm$dd_dist logical name is currently assigned as follows

   "ADM$DD_DIST" = "DUA0:[DD_DIST]" (LNM$PROCESS_TABLE)

Enter Y[ES] if this assignment should be used, N[O] to reassign:
```

DEMOSETUP then calls SETUP.COM (described above in Appendix I.9.3), assigning the current default directory to the logical name ADM$DD (the location of the .ADD files). It also assigns the DEMO application logical name NTB_DATA with the name of the current default disk and directory.

If you want to experiment with the DEMO, perhaps changing the DEMO application, or altering entities and/or relationships in the Data Dictionary, we recommend that you make a copy of the demonstration directory and experiment with the copy.

I.13 Converting Applications to the ADD Environment

This section describes a set of procedures developed to facilitate the conversion of existing ADMINS applications to the ADMINS Data Dictionary (ADD) environment. You might consider converting an application to the ADD environment when substantial enhancement or expansion of the application is being contemplated, and the developer wishes to take advantage of the additional productivity, functionality, and maintainability of the ADD environment. On the other hand, for relatively static mature applications, where major revisions are not likely, conversion to ADD is not necessary, would not produce great benefits, and is therefore not recommended.

I.13.1 Converting Data Files

The procedure to load existing datafiles into the Data Dictionary first acquires information from the DEFs for the datafiles, then lists discrepancies and/or ambiguities in that information, and provides a screen for resolving these problems. Then the "cleaned-up" information is loaded into the Data Dictionary files (the ".ADD" files in your ADM$DD directory.)

This procedure utilizes the following logical name assignments, all must be assigned when the procedure is called.

ADM$DD_DIST	Identifies the ADMINS Data Dictionary distribution directory. Used to locate the files that make up the data file conversion procedures.
ADM$DD_LOAD	Identifies the directory where the conversion procedure will place the temporary files it creates and uses during the loading and correction phases.
ADM$DD	Identifies the directory that contains the Data Dictionary to be updated with information about the files being converted, i.e. where the ".ADD" files are.

I.13.1.1 Loading information from the DEFs

To load information from the DEFs call ADM_DD_LDEF.COM

```
$ @ADM$DD_DIST:ADM_DD_LDEF *.DEF
```

The DEFs to be processed by ADM_DD_LDEF.COM MUST be specified using the "*" wildcard. Some examples:

```
$ @ADM$DD_DIST:ADM_DD_LDEF GL*.DEF              Correct!

$ @ADM$DD_DIST:ADM_DD_LDEF VENDOR.DEF             Wrong!

$ @ADM$DD_DIST:ADM_DD_LDEF LEDGER.DEF,VENDOR.DEF  Wrong!
```

If your organization uses a standard purpose or description line in DEFs (i.e. they all have a line that starts with, for example, "* Purpose:", followed by descriptive text), you can indicate to the procedure that these lines should be used as the file description in the data dictionary. To do this assign the string of characters that your organization uses to mark the standard purpose or description line to the logical name ADM$DD_LOAD_SOURCE_DESCR. For example, the logical name assignment:

```
$ ASSIGN "* Purpose:" ADM$DD_LOAD_SOURCE_DESCR
```

tells the procedure to capture the text "Log results of inspections" from the DEF listed below and use it as the file description in the data dictionary.

```
**********************************************
* System.: Inspectional Services Information System
* Program: RESULT.DEF
* Purpose: Log results of inspections
**********************************************
IS_DATA MAS 1000
*
CLSQN       X99999  KEY1  "Complaint Sequence number"
CLTYPE      X99     KEY2  "Complaint type (inspection type)
INTYPE      I       DKEY3 "Inspection type"
*
INSPDAT     DA             "Date of inspection"
*
   (etc.)
```

ADM_DD_LDEF.COM parses the DEFs specified, then organizes the information from the DEFs to compare entries and determine when the same field name occurs in multiple DEFs. If attributes of each occurrence (field name, field type, description) match exactly, the procedure assumes all are occurrences of the same data element. If the field name and field type of multiple occurrences match, but the descriptions differ,[10] these instances are identified in the procedure's final report (ADM_DD_XARP.REP), for you to resolve in the ADM_DD_XAT1 screen (described below).

ADM_DD_XARP.REP also identifies instances where **the field type differs in multiple occurrences of the same field name.** In these cases **you must change the DEFs** involved so that the field names of the two occurrences are not the same, **then run ADM_DD_LDEF.COM again.**

To correct duplicate entity names (i.e. if two instances have the same field name and data type but different descriptions) call the ADM$DD_DIST:ADM_DD_XAT1 screen and assign the same element number (DD_ID) to both entities (use the DD_ID of either one, it doesn't matter). **Note: whichever file description is listed first in this screen will be the one moved into the dictionary.**

```
ADMINS DATA DICTIONARY              DUPLICATE                24-AUG-90
 (Report ADM_DD_XARP)              FIELD NAMES               Page 1

FIELD_NAME FILE_NAME     DATA_TYPE   DESCRIPTION

CCODE#_CIS RESULT.DEF    I           Complaint code # in ISC100.MAS
           INSPECT.DEF   I           Which (of 4) Prob code in COMPL.MAS

CLSQN      INSPECT.DEF   X99999      Complaint Sequence number
           COMPLOCA.DEF  X99999      Sequential Complaint Number

CLTYPE     INSPECT.DEF   X99         Complaint type (Problem code)
           RESULT.DEF    X99         Complaint type (inspection type)

ISCODE     INSPECT.DEF   X99         Insp. category code
           RESULT.DEF    X99         Inspection Category
```

Figure I10-1: Report ADM_DD_XARP output

10. If you want to exclude the field description from this checking, make the following logical name assignment **before you run ADM_DD_LDEF.COM:**
    *
    $ ASSIGN N ADM$DD_CHECK_DESCR
    *

To access this screen type:

```
$ TRA ADM$DD_DIST:ADM_DD_XAT1
```

The screen appears as follows:

```
*-------------------------------------------------------------------------------*
|ADMINS DATA DICTIONARY              CORRECTION OF DUPLICATE ENTITY NUMBERS  |
*-------------------------------------------------------------------------------*


ENTITY_NAME         RECT  DATA_TYPE    DD_ID   DESCRIPTION
CLTYPE              100   X99          EL0184  Complaint type (inspection type)
CLTYPE              100   X99          EL0178  Complaint type (Problem code)
```

Figure I10-2: Discrepancy - same name, type, different descriptions.

```
*-------------------------------------------------------------------------------*
|ADMINS DATA DICTIONARY              CORRECTION OF DUPLICATE ENTITY NUMBERS  |
*-------------------------------------------------------------------------------*


ENTITY_NAME         RECT  DATA_TYPE    DD_ID   DESCRIPTION
CLTYPE              100   X99          EL0178  Complaint type (Problem code)
CLTYPE              100   X99          EL0178  Complaint type (Problem code)
```

Figure I10-3: Eliminating the discrepancy.

After any discrepancies have been resolved, you are ready to update the data dictionary files. Run ADM_DD_LDEF2.COM to load records describing the fields and files for your application into the data dictionary.

```
$ @ADM$DD_DIST:ADM_DD_LDEF2
```

I.13.1.2 Converting the files

Once the DEFs have been loaded into the Data Dictionary, you'll need to define the files, and MOVE/CONVERT the data. DEFINE the files using the Data Dictionary FILE_contains_ELEMENT relationship screen:

```
NEXT_PG PREV_PG FLD_NO  COPY    COMMIT  DEFINE  FIL_ATR OVRVIEW HELP    MENU
                                        Define Current File Definition
| ADMINS/V32 DATA DICTIONARY       MAINTAIN FILE_contains_ELEMENT RELATIONSHIP |
*-------------------------------------------------------------------------------*
File Name: INSPECT
File Type: MAS        #Records: 1000                        File ID: FI0103

 SEQ FIELD_NAME         FORMAT            KEY/SOR SB OPER    SECONDARY_NAME
   1 DATE_AIS           DA                KEY1
   2 INSPECTOR_AIS      A4                KEY2
   3 INSEQ              I                 KEY3
   4 CLSQN              X99999
   5 CCODE#_CIS         I
   6 CLTYPE             X99
   7 ISCODE             X99
   8 RESULTS            A1
   9 CITATION#          X999999
  10 INTYPE             I
  11
  12
  13
  14
  15
```

Figure I10-4: Defining the file

If the old version of the file is present in the directory where the new version is to be located, the records from the old file will be automatically MOVE/CONVERTed into the new file. If the old version of the file is not present, you'll have to do the MOVE/CONVERT operation yourself.

NOTE

If a discrepancy was discovered in loading the DEFs into the Dictionary that resulted in changing the name of a field, you must use the Dictionary screens to assign a secondary name to that field. That secondary name will relate the new field name in the data dictionary environment to its corresponding (original) name in the old application environment, so that the MOVE/CONVERT operation will be able to find the correct field in the old file, and load its value into the new field name in the new file.

I.13.2 Converting Table Files To Internal Codelists

ADM_DD_CLCNV.COM is used to convert existing ADMINS data files into internal Codelist Tables for use with the ADMINS Data Dictionary. **Keep in mind that internal codelist tables have only one key.** ADM_DD_CLCNV.COM converts one file into the internal codelist table each time it is run.

If your application uses certain files (most likely ".TAB files) exclusively for table lookup, validation checks, etc., these files are fairly static (i.e. not commonly updated by the user), and these files have only a single key, these files can be considered for conversion into internal codelist tables in the Data Dictionary. Also consider that the Data Dictionary allocates 24 bytes for the key, 60 bytes for the description, and 16 bytes for the user action field. If the table file has large number of entries but very small code/description/action field sizes, putting the file in the internal codelist table may not be an efficient use of disk space.

Once you have selected the table files to be converted to internal codelist tables, note the names and data types of the fields in those files that will be converted to the internal codelist's Code, Description, and User Action fields. You will need to provide this information during the procedure (Code will be required, you do not have to specify a field to be used for Description or User Action).

Make entries in the Data Dictionary for each codelist that you wish to convert, using the Data Dictionary Codelist Table Attributes screen:

```
T_VALUES LOOKUP   DESCR    OVERVIEW HELP     MENU
*CL50-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY   User: GINNY                CODELIST ATTRIBUTES |
*-------------------------------------------------------------------------*
Codelist Table Name: EMP_INIT                   DD_ID#: CT0102    INTERNAL

Description........:
Codelist Repository: CL0001  Data Dictionary Internal Codelist
Code Data Format...: A4

                     EXTERNAL CODELISTS ONLY
Codelist Key Field.:
Description Field..:
UAC Field..........:




Added 21-AUG-90   by GINNY           Last changed 24-AUG-90   by GINNY
```

Figure I10-5: Creating the entry for the codelist table.

As each entry is made be careful to note the DD_ID# the Dictionary assigns to that entry. You will have to provide the DD_ID# for each table during the procedure.

After your entries in the Codelist Table Attributes Screen have been completed, call ADM_DD_CLCNV.COM (note: ADM_DD_CLCNV.COM is an ADMINS command file!):

```
$ COM ADM$DD_DIST:ADM_DD_CLCNV
```

You will be prompted for the DD_ID# of the codelist table to be loaded, and the file you will be loading that table with:

```
Enter DD_ID# from Codelist Table:
Conversion Filename (i.e. LEDGER.MAS):
```

Your responses are re-displayed and you must confirm that you want to continue.

```
DD_ID#:  CT0101
Filename: EMP_INIT.TAB

Do you want to continue conversion to Data Dictionary
  Codelist Table (Y OR N)
  ANS: Y
```

Next the Data Dictionary Conversion Screen will appear:

```
        D A T A   D I C T I O N A R Y   C O N V E R S I O N   S C R E E N

DD_ID#: CT0102
Codelist Data Format:

================================================================================

Code:
Convert using (NCAT/FCAT):

Description:
Convert using (NCAT/FCAT):

User Action Code:
Convert using (NCAT/FCAT):

Are you finished with this entry?   (Y/N)
```

Figure I10-6: The Data Dictionary Conversion Screen

You enter the following information:

Codelist Data Format:	Field type from the Codelist Table Attributes Screen (the field type of the key of the file being converted)
Code:	Data file's key field. This field is required.
Description:	Data file field to be used as the description in the internal codelist. This field is optional.
User Action Code:	Data file field to be used as the user action field in the internal codelist. This field is optional.
Convert using (NCAT/ FCAT):	You must specify what method to use to convert the values in the file into the internal table (generally, use NCAT to convert alpha fields and FCAT for integer or decimal fields.)

The following figure shows a completed entry:

```
        D A T A   D I C T I O N A R Y   C O N V E R S I O N   S C R E E N

DD_ID#: CT0102
Codelist Data Format: A4

================================================================================

Code: ISEMP
Convert using (NCAT/FCAT): NCAT

Description: ISEMPN
Convert using (NCAT/FCAT): NCAT

User Action Code: ISCODE
Convert using (NCAT/FCAT): NCAT

Are you finished with this entry? Y (Y/N)
```

Figure I10-7: A completed entry.

When you indicate that you are finished with the entry, the procedure will automatically load the selected fields into the Data Dictionary as an internal codelist table. To specify that this codelist table should automatically generate a Lookup Window for any field that is validated against it, use the Data Dictionary Codelist Table Lookup screen:

```
ATTRIB   T_VALUES OVERVIEW HELP     MENU
*CL52-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                              CODELIST TABLE LOOKUP|
*-------------------------------------------------------------------------------*
Codelist Table Name: EMP_INIT                  DD_ID#: CT0102
Description........: Customer Identification list

Display Fields..: CODE  A4                    DESCRIPTION   User Action Code
Display width...: 8                            20             9
Title...........:
Lookup Heading..:
 Initials    Employee_Name     Insp_Type
Simulated Display
 CCCCCCCC DDDDDDDDDDDDDDDDDDDD UUUUUUUUU
Footing.........:

Lookup Options
 Bound.........: N     (Y/N [N])
 Key Search CAPS: N     (Y/N [N])
Added 24-AUG-90   by GINNY           Last changed 24-AUG-90   by GINNY
```

Figure I10-8: The Codelist Table Lookup screen

Repeat this procedure for each entry you made in the Codelist Table Attributes screen.

I.13.3 Converting the Application

After converting the data files to the Data Dictionary environment, and converting the table files to internal codelist tables, two steps remain to complete the conversion of the application. The first step is to use the Data Dictionary screens to associate files with codelist tables for validation and (if specified) automatic lookup windows. The second step is to change the application code (the TRS and RMS) to utilize the Data Dictionary environment.

I.13.3.1 Associating fields to Codelist Tables

In the Data Element Attributes screen, associate the field to the appropriate codelist table, as shown in the figure below. Note that a Lookup Window is available to view a list of all the codelist tables available in the Dictionary.

```
TEXTATTR HELPTEXT DESCR.   OVERVIEW HELP     MENU
*AT11-------------------------------------------------------------------------*
| ADMINS/V32 DATA DICTIONARY                        DATA ELEMENT ENTRY SCREEN  |
*-------------------------------------------------------------------------------*
Field Name: CLTYPE             DD_User: GINNY            Data Element #: EL0171
                            Prototype:                  Prototype DDID:
Data Format.......: X99
Description.......: Complaint type (Problem code)
Def. display width:
Justification(L/R):
Default Heading 1.:                              Line Label:
Default Heading 2.:

Codelist Table....: PROBCODE
             CT0105 Problem Code
Validation rules.1:
            Line 2:
            Line 3:
Error Message.....:
Options(CAPS/REQU):


Added 21-AUG-90   by ADDLOD             Last changed 24-AUG-90   by GINNY
```

Figure I10-9: Associating a field to a codelist table.

Once the field has been associated with a codelist table, validation and lookup against that table are automatically available in any screen that uses that field (you just have to recompile the TRS).

I.13.3.2 Changing the application code

The listings that follow illustrate some of the ways applications change in the Data Dictionary environment. Note that the Data Dictionary version contains no LINK paragraphs to do validation or to get descriptions for codes; utilizes automatically generated lookup windows in most cases; and contains fewer CHECK statements to support validation of data entries.

The application's maintainability should be improved also, because its validation and lookup functions are centralized, rather than having these functions re-specified for each screen in the application.

```
 *********** BEFORE CONVERSION: RESULT.TRS ***********
* System......: Inspectional Services Information System
* Program.....: RESULT
* Purpose.....: Main Entry for Inspectional Appointment results
RESULT1 IS_DATA:RESULT.MAS 1 IS_PGM:RESULT.RMO NOMSG
INDEX IS_DATA:INSPECT.MAS NO_NULL
INSPDAT DATE_AIS
(etc.)
END
*
LINK IS_DATA:COMPL.MAS W
K  CLSQN
L  CL1DESC_CIS
L  CL2DESC_CIS
(etc.)
END
LINK IS_DATA:PROBCODE.TAB-R
K  CLTYPE
L  PRBDSC
END
LINK IS_DATA:INSP_CAT.TAB-R
K ISCODE
L ISDESC
END
LINK IS_DATA:INSP_TYPE.TAB
K INTYPE
L INTDESC
END
LINK IS_DATA:EMP_INIT.TAB
KC INSPECTOR_AIS
L ISEMP INSP_INIT
L ISEMPN
L ISCODE INISC
END
LINK IS_DATA:RESVAL.TAB
KC RESULTS
L RESULTS L_RESULTS
L RESDESC
END
*
* Append Para inserts inspection appointment
* VINTYPE marks any record as next level (INTYPE + 1)
APPEND IS_DATA:INSPECT.MAS COMMIT X
REINSPDAT       DATE_AIS
LINSPECTOR      INSPECTOR_AIS
LINSEQ          INSEQ
CLSQN
(etc.)
END

                                        WINDOW   13 13 11 30
                                       DISPLAY  RESULTS RESDESC
                                        RETURN   RESULTS
DR L_RESULTS
DR RESDESC                  %BOLD
DR FMTRES/A80               %BOLD+REV+FU
ER CITATION#/X999999        %BOLD
(etc.)

* Append Para inserts RESULT record
```

```
* (One result record for each appointment record)
APPEND IS_DATA:RESULT.MAS CREATRES X
CLSQN
CLTYPE
(etc.)
END
*
V 1HEADER/A21    %BOLD          'Inspectional Services'
V 2HEADER/A30   [4,2,30] %BOLD 'Inspection Results: Complaint '
*
DR ISDESC/A20
DR LINSEQ/I
DR COMMIT/A1
DR CREATRES/A1
ER CLSQN                    %BOLD
ER CLTYPE       [5,16,2]    %BOLD
ER INTYPE       [5,34,2]    %BOLD
*
DR PRBDSC                   %BOLD
DR ISCODE                   %BOLD
DR INTDESC                  %BOLD
ER INSPDAT                  %BOLD
ER INSPECTOR_AIS/A4 [5,71,3] %BOLD %LOOKUP IS_DATA:EMP_INIT.TAB-R
                                      TITLE  Authorized Inspectors
                                    HEADING  Init.   Inspec._Name
                                        DISPLAY  ISEMP    ISEMPN
                                      SELECT   ISCODE EQ ~ISCODE
                                        RETURN   ISEMP
                                        IDENT  INSPECTORS
(etc.)
*
DR INISC/X99
DR INSP_INIT/A4
DR ISEMPN                    %BOLD
DR INSEQ/I
(etc.)
*
ER RESULTS/A1    [15,11,1]  %BOLD %LOOKUP  IS_DATA:RESVAL.TAB-R
                                       TITLE  Results Table
                                    HEADING  Code  Description


ER REINSPDAT/DA   [22,68,6] %BOLD+REV
ER LINSPECTOR/A4  [23,76,4] %BOLD+REV %LOOKUP =INSPECTORS
*
(etc.)
CAPS INSPECTOR_AIS  RESULTS LINSPECTOR
REQUIRE INSPDAT INSPECTOR_AIS RESULTS
*
 C INTYPE NE 1 AND RESULTS EQ 'N'
Improper result code for reinspection: Enter C if problem has been corrected.
C ERR EQ 101
Please enter only in remarks fields when no cause for complaint.
C ERR EQ 102
Enter V(Violation) N(No cause) C(Corrected) or X(Insp. Cancelled)
C ERR EQ 106
 The initials entered are not in the list. (Use FIND to see lookup table)
C ERR EQ 107
 Invalid initials for this inspection type (Use FIND to see lookup table)
 *
BAR 1 WIDTH=10 OPTIONS=VISIBLE
(etc.)
BOX DEFAULT
SCREEN
 BL
BL
DW       1HEADER---------------
DW                          CLSQ-
!Problem Code:       !Insp.Type:          !Inspection  !Inspected by:          !
!PRBDSC--------------!INTD----------------!Date: INSPD-!ISEMP-----------       !
! Location!Street Name=======+==+Number!Ext.!Apt/Unit+=========+===============+
!         !CLSNAME_CIS--------- !CLHO- !CLE-!CLAPT-  !                         !
+=========+====================+======+====+========+                         !
!Complaint!CL1DESC_CIS-------------------------------------------------        !
!Descrip. !CL2DESC_CIS-------------------------------------------------        !
!Owner    !OW_NAME--------------------------------------------------           !
+=========+=============================================================================+
-------------------------------------------------------------------------------FMTRES
```

```
!VRESUL--   RESDESC--------------------- RREG--------------------------------!
!RCITATION------- CITAT-                              RREFCODE-------------  !
+===RXDESC---------------=============================+======================+
!1REPORT-------------------------------------------------!R1SEC--- 1CITSECT-----!
!2REPORT-------------------------------------------------+======================+
!3REPORT-------------------------------------------------!R2SEC--- 2CITSECT-----!
!4REPORT-------------------------------------------------!R1RL--------------    !
!VREMARK-!1REMARK-----------------------------------------!------R2RL            !
!        !2REMARK-------------------------------------!R3RL-------------     !
!Referral to other agencies: REFAGENCY---------- Date: RAGDA-                  !
BRANCHES
*
(etc.)
END


********** AFTER CONVERSION: RESULT.TRS **********
* System......: Inspectional Services Information System
* Program.....: RESULT
* Purpose.....: Main Entry for Inspectional Appointment results
RESULT1 IS_DATA:RESULT.MAS 1 IS_PGM:RESTEST.RMO NOMSG
INDEX IS_DATA:INSPECT.MAS NO_NULL
INSPDAT DATE_AIS
(etc.)
END
LINK IS_DATA:COMPL.MAS W
K  CLSQN
L  CL1DESC_CIS
L  CL2DESC_CIS
(etc.)
END
APPEND IS_DATA:INSPECT.MAS COMMIT X
REINSPDAT       DATE_AIS
LINSPECTOR      INSPECTOR_AIS
LINSEQ          INSEQ
CLSQN
(etc.)
END
APPEND IS_DATA:RESULT.MAS CREATRES X
CLSQN
CLTYPE
(etc.)
V 1HEADER/A21    %BOLD         'Inspectional Services'
V 2HEADER/A30   [4,2,30] %BOLD 'Inspection Results: Complaint '
DR LINSEQ/I
DR COMMIT/A1
DR CREATRES/A1
ER CLSQN                    %BOLD
ER CLTYPE       [5,16,2]    %BOLD
ER INTYPE       [5,34,2]    %BOLD
DR ISCODE                   %BOLD
ER INSPDAT                  %BOLD
V LEMPDD/XAA9999 'CT0101'
V HEMPDD/XAA9999 'CT0101'
V A16ISCODE/A16 NCAT(A16ISCODE,ISCODE)
ER INSPECTOR_AIS/A4 [5,71,3] %BOLD %LOOKUP ADM$DD:ADM_DD_CLIST.ADD
                                    WINDOW 7 30 12 40
                                    TITLE  Authorized Inspectors
                                    HEADING  Init.   Inspec._Name
                                    KEY_RANGE LEMPDD HEMPDD
                                    DISPLAY  CL$CODE/10 CL$DESCR/24
                                    SELECT   CL$UAC EQ ~A16ISCODE
                                    RETURN   CL$CODE
                                    IDENT  INSPECTORS
(etc.)
ER RESULTS/A1    [15,11,1]  %BOLD
DR FMTRES/A80               %BOLD+REV+FU
ER CITATION#/X999999        %BOLD
*
(etc.)
ER REINSPDAT/DA   [22,68,6] %BOLD+REV
ER LINSPECTOR/A4  [23,76,4] %BOLD+REV %LOOKUP =INSPECTORS
*
(etc.)
CAPS INSPECTOR_AIS  RESULTS LINSPECTOR
REQUIRE INSPDAT INSPECTOR_AIS RESULTS
*
```

```
  C INTYPE NE 1 AND RESULTS EQ 'N'
  Improper result code for reinspection: Enter C if problem has been
corrected.
  C ERR EQ 101
  Please enter only in remarks fields when no cause for complaint.
  C ERR EQ 107
   Invalid initials for this inspection type (Use FIND to see lookup table)
  BAR 1 WIDTH=10 OPTIONS=VISIBLE
  *
  (etc.)
  BOX DEFAULT
  SCREEN
  BL
  BL
  DW       1HEADER---------------
  DW                          CLSQ-
  !Problem Code:       !Insp.Type:          !Inspection  !Inspected by:          !
  !D%CLTYP------------!D%INTYP------------!Date: INSPD-!D%INSPECT-------       !
  ! Location!Street Name=======+==+Number!Ext.!Apt/Unit+=========+===============+
  !         !CLSNAME_CIS--------- !CLHO- !CLE-!CLAPT-  !                         !
  +=========+====================+======+====+========+                         !
  !Complaint!CL1DESC_CIS-------------------------------------------------        !
  !Descrip. !CL2DESC_CIS-------------------------------------------------        !
  !Owner    !OW_NAME---------------------------------------------------          !
  +=========+=============================================================================+
  -------------------------------------------------------------------------------FMTRES
  !VRESUL--   D%RESULTS------------------- RREG--------------------------------!
  !RCITATION------- CITAT-                                RREFCODE-------------  !
  +===RXDESC--------------=============================+======================+
  !1REPORT-----------------------------------------------!R1SEC--- 1CITSECT-----!
  !2REPORT-----------------------------------------------+======================+
  !3REPORT-----------------------------------------------!R2SEC--- 2CITSECT-----!
  !4REPORT-----------------------------------------------!R1RL--------------    !
  !VREMARK-!1REMARK---------------------------------------!------R2RL           !
  !        !2REMARK---------------------------------------!R3RL-------------     !
  !Referral to other agencies: REFAGENCY---------- Date: RAGDA-                   !
  BRANCHES
  (etc.)
  END
```

I.14 AdmDDM: Data Dictionary "Batch" Tool

AdmDDM.EXE provides a "batch" environment to populate, update, exchange, and report on the contents of ADMINS Data Dictionaries.

Command line options are:

-ADD=Path	Data Dictionary Path
-INPut=Path	Input File
-OUTput=Path	Output File
-OPTion	File name that contains option specification
-HELP or -?	On-line Help
-IGNORE	Ignore errors and continue processing
-MU	Open ADD files Multi User (default is Single User)

If no INPUT file name is given on the command line, DDM will prompt the user for input. The input can either be DDM instructions, or the name of a file with input instructions preceded with a '@', e.g.

```
AdmDDM> @MY_DIR:addupd.txt
```

If a statement is typed at the `AdmDDM>` prompt, the user is prompted for more input until he types a command or a CR in column one. Usually, however, AdmDDM instructions to generate a new Data Dictionary, or to modify an existing Data Dictionary are provided via an **-INPUT** "batch" file, which is perhaps an edited version of a file generated as **-OUTPUT** from an **EXPORT** command in a previous run of AdmDDM.

The **-OPTion** command line switch

```
-OPTion=PATHNAME
```

allows certain options to be specified, where PATHNAME is the pathname of a text editable file that specifies the options.

The only options currently implemented are:

```
LIST EL <attribute> ALWAYS
```

which will cause the value of attribute to be listed when using the LIST or EXPORT functions even if the values are blank.

<*attribute*> can be any of the EL fields:

```
DESCR FORMAT LABEL HEADER1 HEADER2 WIDTH JUST
```

I.14.1 AdmDDM Commands and Syntax

Many instructions to DDM are of the form

```
COMMAND  ENTITYTYPE=ENTITYNAME  [/ATTRIBUTE[=VALUE] ...]
```

where **COMMAND** is one of the commands described in the following sections,

ENTITYTYPE is one of the dictionary entity types: ELEMENT, PROTOTYPE, FILE, DATAVIEW, CODELIST, CODETABLE or USER

ENTITYNAME is the name of the field, file etc. we want COMMAND to act upon. It may also be a wildcard, e.g. GL* or *XYZ.

/ATTRIBUTE is an attribute name relevant to the entity type.[11]

To exit from AdmDDM, type 'exit' or 'quit'.

I.14.2 ADD, MODIFY, UPDATE, COPY, REMOVE

Use **ADD** to add a new entity to the dictionary.

Use **MODify** to update an existing entity.

Use **UPDate** to update an existing entity, or add it if it does not exist[12].

The COMMAND must start in column 1, while /ATTRIBUTE=VALUE may be given on the same line as the COMMAND, or on the next line indented at least one column. Additional attributes may be given on the same line, or on additional lines. E.g.

```
ADD ELEMENT=NEWFIELD /Description="This is my new field"
       /FORMAT=X99999
       /Label=NewField
```

11.See Section 1.1.2.1
12.The EXPORT command outputs UPDate commands.

COPY copies the attributes of an existing entity to a new entity (it does not copy "file contains field" relationships):

```
COPY ELEMENT=OLDFIELD /NAME=NEWFIELD /LABEL=NewLabel
```

Attributes of the copied element can be altered using the same syntax as ADD, MODIFY and UPDATE.

If an attribute value is to long to easily fit on a line, any line may be continued by typing a \ (backslash) as the last (non white-space) character on the line. E.g.

```
MODIFY FILE=CUSTOMER /descr="This is the \
main customer file"
```

or

```
MODIFY FILE=CUSTOMER /descr = "This is the" \
                              "main customer file"
```

will result in the same value being posted to the DESCRIPTION field.

REMOVE deletes an existing entity:

```
REMOVE FI=TESTFILE
```

I.14.2.1 Entity Types and Attributes

These sections list the attributes for each of the entity types.

I.14.2.1.1 Element and Prototype attributes

ELements and prototypes have the following attributes:

/DESCRiption=	Field description
/FORMat=	ADMINS Data Format
/WIDth=	Field default width
/JUST=	L, R or nothing
/HEADing1=	Heading Line 1
/HEADing2=	Heading Line 2
/LABel=	Field Label
/PROTOtype=	Prototype element name (ELEMENTs only)[a]
/EDITMASK=	Editmask
/CUSTOM1=	Custom Field 1
/CUSTOM2=	Custom Field 2
/CUSTOM3=	Custom Field 3
/CODElist=	Codelist Table Name
/QUALifier=	Codelist Table Qualifier Statement
/VALidate=	Validation logic
/ERRormsg=	Error message on validation error
/OPTions=	Field options (CAPS, REQU, CAP1)

/HELP1=	Help text line 1
/HELP2=	Help text line 2
/HELP3=	Help text line 3
/SUBFIELD =	(Fieldname /start=pos /length=pos) to update subfields, or /REMOVE to remove all subfields

Although the text attributes listed below are valid, they are provided automatically when a TI or TX field is added/modified, and there is normally no need to change them manually:

/INITFILE=	# (Valid code from the ADM$DD_TEXT_INITFILE codetable)
/RULER=	# (Valid code from the ADM_DD_TEXT_RULERS codetable)
/TEXTFORMAT =	1 or TED if internal TED WP format 2 or RTF if internal WinTed (AdmTed) format (Win32 only) 3 or WORD if Microsoft WinWord (Win32 only) 10 or TED if external TED WP format 11 or RTF if external WinTed format (Win32 only) (TED WP format will automatically be changed to RTF format the first time the editor is called up in Win32)
/LANGUAGE =	# (Valid code from the ADM$DD_TEXT_LANGUAGES codetable)
/LINEWIDTH =	# (Maximum allowed line width in characters
/WORDFILE =	# (Valid code from the ADM$DD_TEXT_WORD_FILE codetable)
/STRUCTUREFILE=	# (Valid code from the ADM$DD_TEXT_STRUCTURE codetable)
/OWNERACCESS=	/Access code ("", R, W, RW)
/GROUPACCESS=	Access code ("", R, W, RW)
/WORLDACCESS=	Access code ("", R, W, RW)

a.When an element (field) is updated with a prototype, all fields are defaulted to the prototype value. Thus any overrides must appear after the prototype value. E.g. the following is incorrect:

```
update Element=FEBAMT
    /descr     = "February Amount"
    /prototype = AMOUNT
```

In this case the description will be wiped out by the prototype update. The correct way to do it is:

```
update Element=FEBAMT
    /prototype = AMOUNT
    /descr     = "February Amount"
```

I.14.2.1.2 File attributes

FILE attributes are:

/DESCRiption=	"File description"
/DIRECtory=	Directory path (or logical name)
/TYPE=	File type (e.g. MAS, IDX, etc.)
/NRECords=	Number of records
/LOGRecords=	Number of Log File records
/IXONLY=	Yes or No
/DEFOPT=	R RI I X XI or ' '
R	DEF/REDEF if file already exists
I	DEF/INIT if file does not exist
X	Delete file if it exist before defining new
/SELECT=	"Selection statement for file"
/FIELD=	FieldName [/REMove || /DELete] || [/KEYn || /DKEYn || /ASCn || /DESCn || /POS=n || /AFTER=FieldName || /REPLACE=FieldName] /OPERator=[AVG || MAX || MIN] /SIGnificantbytes=n /SECondaryname=SecondaryFieldName If more than the Field Name is given, all attributes must be enclosed in paranthesis, e.g. /field=(MYFIELD /key1 /secname=SECNAME) If only a field name is given, e.g. /field=MYFIELD the field is added at the end of the relationship. /FIELD=/REMOVE will remove all fields from the file. (NYI)
/INDEX=	(Index# IndexName KeyFld1 [KeyFld2]...) specifies alternate indexes, e.g.: **/index = (1 'Owner Name' LNAME FNAME)** **/index = (2 'byOrg' ORGCODE)**

I.14.2.1.3 Dataview attributes

DATAVIEW attributes are:

/DESCRiption=	"Data View description"

I.14.2.1.4 Codelist attributes

CODELIST attributes are:

/DESCRiption=	"Codelist description"
/FILE=	"File name where codelist recides"

I.14.2.1.5 Codetable attributes

CODETABLE attributes are:

/DESCRiption=	Codelist Table description
/TYPE=	INTERNAL (if Internal Codelist)
/DEPOsitory=	Codelist name (if External Codelist)
/FORMat=	ADMINS Data Format of code

/UPDATE=	(CODE, DESCRIPTION,UAC)
	Provide values to update an existing entry in the codelist table, or add if it does not exist, e.g.:
	`/update=("1001", "Accounting", "`
	`")`
	`/update=("2001", "Banking", " ")`
	`/update=("3001", "Clerical", "`
	`")`
	`/update=("4001", "Domestic", "`
	`")`

/ADD=	(CODE, DESCRIPTION,UAC) to add a new code, generates an error if the code exists.
/MODify=	(CODE, DESCRIPTION,UAC) to update an existing code, generates an error if it does not exist.
/REMove=	CODE to removee an existing code.
/LK_TITLe =	Lookup Title
/LK_HEADing=	Codetable Lookup Heading
/LK_FOOTing=	Codetable Lookup Footing
/LK_CODELength=	Display length of code
/LK_DESCRLength=	Display length of description
/LK_UACLength=	Display length of UAC
/LK_OPTion=	[CAPS] [BOUND]

I.14.3 LIST and EXPORT

LIST and EXPORT are similar, they list attributes for existing entities. LIST reports the information while EXPORT creates AdmDDM commands suitable to UPDATE another dictionary.

For example:

```
LIST FILE=GL*
```

will list all files starting with GL, and

```
EXPORT FILE=GL*
```

will "export" all files.

LIST and EXPORT have a /SINCE modifier to limit the entities listed or output to those changed or added since a particular date, or TODAY

```
LIST FILE=GL* /SINCE=TODAY
EXPORT PE=* /SINCE=1-DEC
```

/SINCE must be preceded by at least one whitespace character.

The **LIST** and **EXPORT** commands also accept

```
/VALues[=ONLY]
```

to list/export all the CodeTable values for an internal codelist, e.g.

```
EXPORT CT=mytable /values
```

If you use **/values=only** no other attributes will be listed.

I.14.4 VERIFY, WHEREUSED and DEFINE

Use **VERify** to confirm that the actual file in use reflects the relation described in the dictionary.

```
VERIFY FI=GL*
```

will verify all files starting with GL.

```
VERIFY FI=GL*/FULL
```

will verify all files starting with GL, and give a full listing of the differences.

WHEreused (or just **USEd**) lists where an entity is used (no wildcards allowed). Use **WHEREUSED/CSV** to get the output in CSV format.

```
WHEREUSED EL=ORGCODE
```

or

```
USED/CSV EL=ORGCODE
```

Use **DEFine** to create a .DEF for a file:

```
DEF FI=GLTRANSACTIONS
```

will create a GLTRANSACTIONS.def for the GLTRANSACTIONS file. If the logical names DDM_SYSTEM and DDM_SUBSYSTEM are assigned those values will be included as part of the documentation in the .DEF.

I.14.5 CSV and NOREF

The CSV command in AdmDDM provides a flexible tool for producing Data Dictionary information in CSV format. The general syntax is:

```
CSV ENTITY=wildcard [ /since=date ]
    /fields=(field1 field2 field3 … )
```

E.g.

```
CSV EL=*
    /fields=(ddid name format descr)
```

will list the Data Dictionary ID, Name, Format and Description for all elements (fields) in the Data Dictionary in the format:

```
"EL0105","ZIP","X99999","ZIP Code"
```

If a text file, e.g. FIELD_LIST.DDM, exists that contains the above CSV command:

then the following command:

```
AdmDDM /inp=field_list.ddm /out=fields.csv
```

will create the file fields.csv with the information asked for.

Another example:

```
CSV FI=GL* /SINCE=1-DEC /FIELDS=(DDID NAME TYPE DESCR)
```

will produce a CSV output containing DDID, File Name, File Type and Description for all files starting with GL added since December 1.

The **/FIELDS** qualifier allows you to list attributes for all the fields in a given file relationship.

```
CSV FI=MYFILE/FIELDS
/HEADING=(Element Name,Description,Format,Codelist)
/FIELDS=(name descr format ct_name)
```

will create a CSV file with the NAME, DESCR, FORMAT and CT_NAME for every field in the file.

Use **NOREF** to list (in CSV format) entities which are not referenced anywhere (e.g. fields not being used in any files, files without fields etc).

```
NOREF EL=*
```

will list DDID, Name, Description, AddedBy and AddedDate of all data elements (fields) in the dictionary which are not being used anywhere.

The CSV and NOREF commands recognize the following field names:

ELEMENTS and PROTOTYPES (EL and PE):

DDID	"Data Dictionary ID for entity"
NAME	"Name of entity"
FORMAT	"Data format (type)"
DESCR	"Description of entity"
PE_ID	"Dictionary ID of Prototype"
PE_NAME	"Prototype name"

LABEL	"Line label"
HEADING1	"1st line of heading"
HEADING2	"2nd line of heading"
WIDTH	"Display width"
JUST	"Display justification"
CT_ID	"Dictionary ID of codelist Table"
CT_NAME	"Name of Codelist table"
CT_ERRMSG	"Error message if not in Codelist Table"
EDITMASK	"Editmask"
SUBFIELDS	"List of subfields"
ERRMSG	"Error message if not passing validation logic"
OPTIONS	"Options (CAPS etc.)"
MICS	"Miscellaneous"
ADDEDBY	"Entity added by"
ADDEDDATE	"Date entity was added"
CHANGEDBY	"Last change by"
CHANGEDDATE	"Date of last change"

FILES (FI):

DDID	"Data Dictionary ID for entity"
NAME	"Name of entity"
TYPE	"File type (e.g. MAS, TAB etc.)"
DESCR	"Description of entity"
DIRECTORY	"Directory (may be logical name) of file"
NRECS	"Number of records"
LOGRECS	"Number of log records"
IXONLY	"Y if Index Only file"
DEFOPT	"Define Options"
INDEX	"List Alternate Indices"

SELECT1	"1st line of selection"
SELECT2	"2nd line of selection"
SELECT3	"3rd line of selection"
ADDEDBY	"Entity added by"
ADDEDDATE	"Date entity was added"
CHANGEDBY	"Last change by"
CHANGEDDATE	"Date of last change"

CODELISTS (CL):

DDID	"Data Dictionary ID for entity"
NAME	"Name of entity"
TYPE	"Type (Internal or External)"
DESCR	"Description of entity"
FI_ID	"Dictionary id of repository"
FI_NAME	"Name of repository"
ADDEDBY	"Entity added by"
ADDEDDATE	"Date entity was added"
CHANGEDBY	"Last change by"
CHANGEDDATE	"Date of last change"

CODELIST TABLES (CT):

DDID	"Data Dictionary ID for entity"
NAME	"Name of entity"
DESCR	"Description of entity"
FORMAT	"Data format (type) of code"
CL_ID	"Data Dictionary id of codelist"
CL_NAME	"Name of Codelist"
CODE_ID	"Data Dictionary id of Code field"
CODE_NAME	"Name of Code field"

DESC_ID	"Data Dictionary id of Description field"
CDESC_NAME	"Name of Description field"
UACT_ID	"Data Dictionary id of UAC field"
UACT_NAME	"Name of UAC field"
ADDEDBY	"Entity added by"
ADDEDDATE	"Date entity was added"
CHANGEDBY	"Last change by"
CHANGEDDATE	"Date of last change"

Appendix J:The TED Text Editor

This Appendix describes the TED text editor, which is automatically called when a TRANS text window is opened for an ADMINS internal (TInn) or external (TXnn) text field.

J.1 AdmTed

AdmTed.exe is the Win32 version of the ADMINS OpenVMS TED editor for internal text. It is a full WYSIWIG graphical editor using the Rich Edit Control (with enhancements) defined in the MFC interface.

AdmTed is the default[1] environment for editing internal text. It is automatically called whenever an ADMINS internal text field is to be displayed or edited in TRANS. It has all the editing capabilities of WORDPAD and some enhancements specific to AdmTed.

The command line switch /t=<window title> can be used to specify the window title of the AdmTed session. Use quotes if the specified Title has embedded blanks, for example:

```
admted /t="Edit Report"  sample.rep
```

J.2 TED Function Keys

TED has many function keys pre-defined to perform various text editing operations. All of these function keys are "generic" keys, i.e. the keys are defined as performing a specified function, but the actual keystroke necessary to perform that function may be designated by the user.[2] A default keystroke (for VT and workstation console mode) is defined for each function.

The following table describes TED's function keys, grouped by general functional area.

GENERAL

1. You can designate another editor, if desired.
2. Redefinition of function keys is described in Appendix J.6 "The TED.ENV File".

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
GOLD	PF1	F1	Used to invoke certain two keystroke functions, e.g. GOLD+CUT will paste the pastebuffer into your text at the cursor position. When redefining TED keystrokes, any TED function keymay be defined by a combination of the TED GOLD key, and any other keystroke. As a general rule, most functions may be performed n times by typing GOLD+nnn in front of the function keystroke, where nnn is a decimal number. E.g. the keystrokes GOLD 12 * cause twelve * characters to be inserted, and the keystrokes `GOLD 5 ->` move the cursor right five columns. If the GOLD+nnn keystrokes are typed in front of a function that does not support multiple occurrences, the GOLD+nnn keystrokes are ignored.
HELP	HELP	F2	Invokes TED's on-line HELP screens.

CURSOR MOVEMENT

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
UPARROW	UPAR	Same	Moves the cursor to the line above the current line.
DOWNARROW	DOWN	Same	Moves the cursor to the line below the current line.
RIGHTARROW	RIGH	Same	Moves the cursor one column to the right.
LEFT ARROW	LEFT	Same	Moves the cursor one column to the left.
PGUP	PREV	Same	Moves one screen upwards in the file. The number of lines moved will depend on the length of your display window.
PGDOWN	NEXT	Same	Moves one screen downwards in the file. The number of lines moved will depend on the length of your display window.
NEXT_WORD	KP_1	Same	Moves the cursor to the beginning of the next word. If pressed at end of line, the cursor will move to the beginning of the next line.
PREV_WORD	KP_4	Same	Moves the cursor to the front of the previous word. If pressed when the cursor is in column one, the cursor will move to the end of the previous line.
END_WORD	KP_8	Same	Moves the cursor to the end of the current, or next word.
BEG_LINE	F9	Same	Moves the cursor to the beginning of the line. If pressed in column one, the cursor will move to the beginning of the previous line.
END_LINE	KP_2	Same	Moves the cursor to the end of the line, or to the end of the next line if it already is at the end of the current line.
NEXT_LINE	KP_0	Same	Moves the cursor to the beginning of the next line.
HOME	FIND	HOME	Brings the cursor to the frist printable character on the line. If pressed twice, it will bring the cursor to the beginning of the line if not already there.
TOP_SCREEN	GOLD+UPAR	Same	Brings the cursor to the top of the displayed screen.
BOT_SCREEN	GOLD +DOWN	Same	Brings the cursor to the end of the displayed screen.
NEXT_PARA	KP_7	Same	Moves the cursor to the beginning of the next paragraph.
PREV_PARA	GOLD+KP_7	Same	Moves the cursor to the beginning of the previous paragraph.

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
SCROLL_RIGHT	GOLD+RIGH	Same	Moves the viewing window over to the right.
SCROLL_LEFT	GOLD+LEFT	Same	Moves the viewing window over to the left.
TOP_OF_FILE	GOLD+T	Same	Moves the cursor to the top of the file.
END_OF_FILE	GOLD+B	Same	Moves the cursor to the end of the file.
GOTO_LINE	GOLD+L	Same	Prompts for a line number. When entered, puts the cursor at that line number.

CUT AND PASTE

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
SELECT	SELECT	KP_.	Toggles select on and off. The word SEL will be displayed on the status line when SELECT is active. When you move about in the file with SELECT active, all text in the selected area will be displayed in reverse video.
CUT	KP_6	Same	Cuts the selected area from your text and puts it in the "paste buffer".
COPY	GOLD+K	Same	Copies the selected area in your text into the paste buffer, but leaves text where it was.
PASTE	GOLD+KP_6	Same	Pastes the contents of the paste buffer into your text at the current cursor position.

LOCATE TEXT

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
SEARCH	PF3	F3	Searches for a specified text string.

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
ENTER_SEARCH	GOLD+F3	GOLD+F3	Prompts for a text string to search for. If you end the search string with RETURN, TED will search for the specified text from the cursor though to the end of the file. If you end the search text with UPARROW, the search will be from the cursor position towards the top of the file. If the search text is terminated by BEG_LINE, TED will only search for the specified string at the beginning of a line. If the search text is terminated by SELECT, TED prompts for a substitute string, i.e. a string to replace the specified search text, if it is found. Entering GOLD+nnn in front of the SELECT keystroke will cause the search text to be replaced by the substitute string nnn times. Initially, SEARCH is case insensitive. Pressing CHANGE_CASE before terminating the search text tells TED that the search is going to be case sensitive, i.e. TED will look for an exact match for the search string you entered. (CHANGE_CASE toggles between case sensitivity and case insensitivity).
REV_SEARCH	KP_5	Same	Reverses the direction of search.

TEXT DELETION AND RESTORATION

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
DEL_CHAR	KP_1	DELETE	Deletes the character under the cursor.
ERASE_CHAR	<x]	BACKSPACE	Deletes the character in front of the cursor. If the ERASE_CHAR key is pressed when the cursor is in column one, the current line will be joined with the previous line, provided the joined line will not overflow the maximum line length in effect.
UNDEL_CHAR	GOLD+KP_,	GOLD+DELETE	"Undeletes", or restores, the last character deleted using the DEL_CHAR or ERASE_CHAR keystroke.
DEL_WORD	KP_-	Same	Deletes from the cursor position up to the next word, including any white spaces between the words.

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
ERASE_WORD	CT_J	KP_/	Deletes the word to the immediate left of the cursor.
UNDEL_WORD	GOLD+KP_-	Same	"Undeletes", or restores, the last word deleted using the DEL_WORD keystroke.
DEL_LINE	PF4	F4	Deletes the current line, regardless of where the cursor is on the line.
ERASE_LINE	CT_U	Same	Deletes all characters on the line in front of the cursor.
UNDEL_LINE	GOLD+PF4	GOLD+F4	"Undeletes", or restores, the last line deleted using the DEL_LINE or ERASE_LINE keystroke.
DEL_ENDLINE	GOLD+KP_2	Same	Deletes all characters from the cursor position to the end of the line.
INSERT_LINE	GOLD+KP_0	Same	Inserts a new line at the position of the cursor.

SPECIAL EDITING KEYS

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
INS_OVR	INSERT	Same	Toggles insert and overwrite mode.
CHANGE_CASE	GOLD+KP_1	Same	Toggles the case of the character under the cursor or the characters in the select range. In a search operation, the key is used to toggle case sensitivity within the search.
SWAP_CHAR	GOLD+X	Same	Swaps, or reverses the order, of the character under the cursor, and the character next to it.
CENTER	GOLD+C	Same	Centers the text of the current line.

INS_GRID	GOLD+G	Same	Inserts a grid above the current line. The grid consists of two lines showing the column numbers, and looks like this:
			`1 2 3 ...`
			`12345678901234567890123456789012345678901234567890123 ...`
			The grid is actually inserted in the text, so make sure you delete it if you only want it temporarily to measure up some text.

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
LINE_DRAW	GOLD +INSERT	Same	Toggles between normal text mode and line draw mode. Under line draw mode the arrow keys are used to draw boxes and lines. In line draw mode, the left "<" or right ">" angle bracket characters may be used to move the cursor left or right without drawing a line, and the L and SPACEBAR keys may be used to move the cursor up and down. The Delete key erases the character under the cursor. Any other key exits from line drawing mode.
FRAME	GOLD+F	Same	Toggles between normal text mode, and frame mode where the arrow keys may be used to draw a box using the frame character.
SPEC_INS	GOLD+KP_3	Same	Prompts for: Special character to insert: _ Allows insertion of any special character into the text. Insert a special character by typing its decimal value, e.g. 27 for <ESC>. If the character is nonprintable, a mnemonic will be displayed instead of the character.
EXT_CHAR	KP_3	Same	Select external character from table. Displays the following small subset of the 8 bit characters: Æ æ Ø ø Å å Ä ä Ö ö Ü ü ½ ¼ ² ³ ± « » ¡ ¿ Ñ ñ Use the arrow keys to move the cursor to the desired character, and then press Select to insert the character in your text. 8-bit characters not available from EXT_CHAR may be entered using SPEC_INS.

MISCELLANEOUS KEYS

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
REFRESH	CT_W	Same	Repaints the screen.
EXIT	CT_Z	Same	Leaves screen editing mode and enters command mode.
REVEAL_CODE	GOLD+F9	Same	Displays hidden codes (i.e., underline, bold, fonts, etc.) on the current line.
EXPLAIN_KEY	GOLD+?	Same	Prompts: "Keystroke to explain:" When keystroke is entered, displays TED function invoked by that keystroke.

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
FONT	CT_F	Same	Displays available fonts in a Font Selection Window. Fonts must be in ADM$STYLE table in order to be used.
DISCR_HYPHEN	GOLD+-	Same	Inserts a discretionary hyphen ("hidden" hyphen) in text. Will hyphenate word at loctaion designated instead of wrapping whole word to the next line (e.g. if you are typing a long word and are near the end of a press GOLD+- where you want the word hyphenated).
ASCII_TABLE	GOLD+A	Same	Displays all ASCII characters with decimal codes between 0 and 127 (7 bit characters) on one screen, and all characters with decimal values between 128 and 255 (8 bit characters) on a second screen. Decimal and hexadecimal values as well as printed symbols are shown. When in screen one, CT_Z or UPARROW will take you back to your editing session, any other character will display screen two. When in screen two, UPARROW will take you back to screen one, any other keystroke will take you back to your editing session.
COMMAND	GOLD+HELP	GOLD+F2	Leaves screen editing mode and enters command mode.
STATUS	GOLD+S	Same	Displays statistics about the editing buffers currently in use.

HIGHLIGHTING

TED can highlight (e.g. boldface) portions of the text by using special highlighting keystrokes. Highlighting may be specified by a select range terminated by a highlighting keystroke, or by a highlighting keystroke followed by the characters to be highlighted. In the latter case, a right-arrow keystroke may be used to bypass the end of highlighting mark. The highlighting keystrokes are:

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
BOLD	CT_B	Same	Changes text to Boldface font.
UNDERLINE	CT_L	Same	Underlines.
ITALIC	CT_K	Same	Changes text to Italic font. Not all printers, or print fonts, support italic characters.

SPECIAL WORD PROCESSING KEYS

Function Key Name	Default VT Keystroke	Default Workstation Keystroke	Description
ERASE _CONTROL	GOLD+E	Same	Erases the control line (e.g. ruler, page break or heading indicator) on the line above the cursor. Asks for confirmation before it erases the control line.
INS_HEADING	GOLD+H	Same	Inserts a Heading. Two control lines delimiting the heading are inserted into the text. Insert the heading text you want between these control lines. The heading text will be printed at the top of every page. See Appendix J.8 for more information on headings.
INS_FOOTING	GOLD+J	Same	Inserts a Footing. Footings work like headings, but are inserted at the bottom of every page.
INDENT	GOLD+I	Same	Forces automatic indentation to a given column. If automatic indentation is set using INDENT spaces are automatically inserted in the file to indent to the specified after each <C.R.>.
NORMALIZE	GOLD+N	Same	"Normalizes" the text between rulers, provided the right ruler character is J or W.
PRINTER_CTRL	GOLD+O	Same	Displays and allows you to alter the printer control fields, e.g. set the number of characters printed per inch, the page length, etc.
PAGE_MARKER	GOLD+P	Same	Inserts a Page Marker above the cursor.
RULER	GOLD+R	Same	Allows you to edit a new ruler.
SPELL_CHECKER	GOLD+F12	Same	Check spelling in text.
LANGUAGE	GOLD+F11	Same	Select language for checking spelling from menu of available languages.
DIALECT	GOLD+F10	Same	Select dialect for checking spelling from menu of dialects available for the active language.

J.3 Rulers

A **ruler** is a virtual line that contains margin, wrapping and tab stop information for the text lines that follow. Rulers look like this:

L------M----------T------------------------T-------W

To edit a ruler press the **RULER** key. The current ruler will be displayed at the message line. You may change the ruler (e.g. set new left and right margins, set tab positions etc.) by editing it using the following characters.

-	No special editing rules apply in this position. Any character is allowed.
L	**Left Margin**. A Carriage Return at the end of a line will always create a new line starting at this position.
M	**Auto-wrap left margin**. If a new line is created because of an automatic wrapping, the new line will start in this position. M may appear either left or right of L.
R	**Right Margin**, no auto-wrap
W	**Right Margin**, automatic word-wrapping
J	Right margin, automatic word-wrapping with right justified lines printed.
T	**Tab stop,** i.e. a TAB keystroke in front of this position on a line in the text will bring the cursor to this position. Any other character than the '-' on the ruler line is considered a tabular position.
>	Right justify at tab stop. If there is room to the left, any text typed at this tab stop will be right justified.
.	Decimal tab stop. Numbers typed at tab stop will have their decimal point at this position.

The following keys are used to edit the ruler:

SPACE	Acts the same as right arrow, but will also wipe out T characters under it.
TAB	Move to next tabular position.
HOME	Brings cursor to the **left** margin.
END_LINE	Brings cursor to the **right** margin.
->	Move **right** one position.
<-	Move **left** one position.
EXIT	Cancel editing of ruler, and exit to text editing mode.
<CR>	Store edited ruler. If no changes were made to the ruler, the old ruler is kept.

When you are finished editing the new ruler, press **<CR>** to store it. The ruler is stored immediately above the line at the cursor when the **RULER** key was pressed. If there is a ruler present at this position, it will be replaced with the edited ruler. If no ruler is present at this position, the edited ruler will be inserted at this position.

If no changes have been made to the ruler (i.e. the edited ruler matches the current ruler), the old ruler is kept.

If, after making changes, you decide not to store the edited ruler, the **EXIT** key will cancel the changes and bring you back to text editing mode at the position where ruler editing mode was invoked.

J.4 Checking Spelling

TED can check your document for spelling errors.[3] If you press the SPELL_CHECKER key, the "Check:" prompt will appear at the bottom of the display:

```
Check: x=Exit 1=Word 2=Screen 3=Document 4=Reset skips [x]:
```

Reply "x" to exit the spelling checker immediately, without any spelling checking done. "x" is the default, so if you simply press RETURN you will exit the spelling checker. Reply "1" to check the word the cursor is currently on. Reply "2" to check the text currently displayed on the screen. Reply "3" to check the entire document. Reply "4" to eliminate the list of words you have told TED to ignore (see "Not found:" option 2, "Skip", described below).

If a word in the area being checked is not found in the active dictionaries TED prompts "Not found:" to determine how you want to handle this word:

```
Not found: 1=Skip once 2=Skip 3=Add 4=Edit 5=Exit:
```

Reply "1" if you want to leave the word as it is, but you want TED to let you know if the same spelling is found again. "1" is the default response, i.e. if you press RETURN TED acts as if you pressed 1. Reply "2" if you want to leave the word as it is, and you want TED to ignore this word if it is encountered again in this session.[4] Reply "3" if you want to add this word to your Personal Dictionary (see Appendix J.4.2 "Spelling Checker Personal Dictionary"). Reply "4" if you want to edit the spelling of the word. Reply "5" if you want to exit the spelling checker.

3. TED's spelling checking uses the Houghton Mifflin Company's International Correctspell product. On OpenVMS Systems TED expects the spelling checker image file (ICSPL.EXE) to be in the directory assigned to the logical name ADM$DIST. If ICSPL.EXE is not in the ADM$DIST directory you must tell TED where it is by assigning its file specification to the logical name TED$SPELL, e.g.: "assign mydisk:icspl.exe ted$spell"
4. For TED in TRANS the word will be skipped for all TED sessions within the TRANS session. If you want TED to stop ignoring words you told it to ignore earlier in the same session, use "Check:" option 4, "Reset Skips".

The spelling checker tries to provide suggested spellings for the word it cannot find. When it has suggestions, TED displays them in a list, each suggestion preceded by a letter. The following example shows a list TED might display if the word "tipe" appeared in your document"

```
a: tie
b: tip
c: type
```

To select one of the suggestions, enter the letter that precedes it (in the above example you would enter "c" to replace "tipe" with "type".

J.4.1 Language Support

Spelling checking is done in "American English" by default.[5] Checking can be done in any of the following languages and dialects:

```
Language     Dialects
--------     -------
Danish
Dutch
English*     american*, ise_english, ize_english, australian
Finnish
French
German
Italian
Norwegian    bokmal*, nynorsk
Portuguese   iberian*, brazilian
Spanish
Swedish
```

In the above list the default language and default dialects for each language are indicated by asterisks (*).

To use another language assign the name of the language to the logical name ADM$ICS_LANG. For languages with more than one dialect the default dialect is used unless the name of another dialect is assigned to the logical name ADM$ICS_DIALECT. For example, to specify Brazilian Portuguese make the following logical name assignments:

```
Open VMS:
    $ assign portuguese adm$ics_lang
    $ assign brazilian adm$ics_dialect

UNIX:
    $ lcr adm_ics_lang portuguese
    $ lcr adm_ics_dialect brazilian
```

5. ADMINS is normally shipped with the English dictionary only. To instead receive up to three languages (e.g. English and two others) send us a written request identifying which languages you would like to have shipped with your distribution tape.

During the TED session, you can change the language by picking from a menu of the languages available[6] by pressing the LANGUAGE keystroke. In the menu move the cursor to the language you want and press SELECT. To leave the menu without making a choice press EXIT.

Similarly, for multi-dialect languages, you can change the dialect by picking from a menu of the dialects available by pressing the DIALECT keystroke.

The language and dialect can also be changed at the "Command:" prompt or in the TED environment file via the **language** and **dialect** keywords. The following lines specify Brazilian Portuguese using keywords:

```
language=portuguese
dialect=brazilian
```

The dictionary language database files have names in the form admxxx##.ics (xxx indicates the language and ## the dialect, i.e. admeng01.ics, admnor01.ics, etc.) and reside in the ADM$DIST directory by default. To tell TED to use language database files in another directory assign the path specification to the logical name ADM$ICS_PATH.

J.4.2 Spelling Checker Personal Dictionary

Professions, organizations, and topic areas have unique nomenclature, jargon acronyms, etc., that occur normally in documents but would not be found in any generalized dictionary. To accommodate these special situations TED allows you to customize spelling checking by adding words to a "personal dictionary". Before TED's spelling checking presents a word as possibly misspelled it will check to see if it is in your personal dictionary.

To use a personal dictionary, you must first create a file to hold it, and assign its full path specification to the logical name ADM$ICS_PERS_DICT.

Create the file with any text editor. You can leave the file empty and build your personal dictionary by adding words to it via the spelling checker "Add" option, or you can enter a list of words, one per line, to start your personal dictionary, as in the following example:

```
      |note leading spaces
      || | |
      |V V V
column|123456789...
------+---------------
      | 0 0 ADMINS
      | 0 0 TRANS
      | 0 0 ADBS
      | 0 0 TED
      |
      | (etc.)
```

6. The languages available and displayed in the menu are determined by the the setting of the "languages" keyword in the TED$ENV file. For example, if you have the English, French, and German dictionaries put the line "languages=english, french, german" in your TED environment file to have all three languages displayed in the languages menu.

Each word begins in column 6 and is preceded by the sequence[7] "space/zero/space/zero/space", which begins in column 1.

Assign the full path name of the file you create to the logical name ADM$ICS_PERS_DICT, e.g.:

```
$ ASSIGN DISK:[MYDIR]MY.DICT ADM$ICS_PERS_DICT  !OpenVMS

$ lcr ADM_ICS_PERS_DICT /home/mydir/my.dict     #UNIX
```

Whenever the spelling checker finds a word that cannot be verified against an active dictionary, you can choose "Add" to include the word in your personal dictionary. Once a word is added it will be found for the remainder of the TED session, and in any subsequent session when that same personal dictionary file is active.

You may have two personal dictionary files active at a time. This allows organizations to have a company-wide personal dictionary for terms commonly used throughout the organization, while also maintaining individual personal dictionaries for individual users. To enable a second personal dictionary create a personal dictionary file as above, or copy a personal dictionary file to a new name, or combine several personal dictionary files into a new file; and then assign the name of that file to the logical name ADM$ICS_PERS_DICT_1, e.g.:

```
$ ASSIGN DISK:[CENTRAL]ORG.DICT ADM$ICS_PERS_DICT_1   !OpenVMS

$ lcr ADM_ICS_PERS_DICT_1 /home/central/org.dict      #UNIX
```

When two personal dictionary files are active, both are checked before a word is presented as possibly misspelled; but **new "Add" words are always added to the "individual" file** (the one identified by the logical name ADM$ICS_PERS_DICT).

J.5 Command Mode (Screen)

Exiting from **Screen Edit Mode (via EXIT** or **COMMAND)** puts TED in **Command Mode.** In command mode TED prompts on the bottom line of the display:

Command:

The following commands are available in Command Mode:

EXit	Save the buffers and exit. The CURRENT buffer will always be written to disk, and TED will ask you if you want to save any other buffers that have been changed.
Quit	Exit from TED without saving the buffers to the disk. If you have made changes to any of the text buffers, TED will ask you if you want to save them.
QP	Exits from TED without saving the buffers to the disk and saves position in file.

7. The two zeros that precede the entries are internal codes used by the spelling checker.

SAve [filename]

Write the current buffer to the disk, but do not exit from TED. All your text buffers will be available for continued editing after a save. When a buffer other than the main buffer is active, you can use

```
SAVE file_name
```

to write the contents of that buffer as a text file with that file_name.

WRite [filename]

Write the current buffer to the disk, and empty the buffer. (the **main** buffer is not written or emptied). Use this command to free a text buffer you are finished editing, and you either want to free up the memory occupied by the buffer, or you want to make the buffer available for another text file to edit. Use

```
WRite file_name
```

to write the buffer under a different name, or to write the main buffer as a text file with that file_name.

INclude file_name [=buf_name]

Loads text file **file_name** into your current text buffer, at the cursor location. If the optional "=buf_name" is used, TED loads text file **file_name** into buffer **buf_name,** and enters Screen Edit mode for that buffer. If the file is not found, TED will ask you if you want to create the new buffer **buf_name.**

INcludeRO file_name [=buf_name]

(can be abbreviated to INRO) Includes text into a buffer limited to "read-only" access. Read-only buffers cannot be altered or written to disk. If you use INRO to include a file into a buffer that already includes text the entire buffer is made read-only. Any changes made to the text not saved previously will be lost.

If you use INRO to include a file into a buffer that is not already read-only TED prompts for confirmation:

This will make buffer 'MAIN' Read Only. OK to continue?

LOCK

Write-locks the text above the current cursor position. Locked text is displayed but cannot be altered. TED prompts for confirmation whenever you use LOCK.

Lock text above line 327?

Reply "Y" to confirm that the text above the current cursor position should be locked. This feature is especially useful to prevent alteration of the existing portions of a document stored in TInn fields, while permitting additions.

Note that **once text is locked it cannot be unlocked. Locks cannot** be removed or relocated closer to the beginning of the file. You may, however, use **LOCK** to relocate a lock toward the end of the file.

=

Lists the names of all your text buffers currently in use.

=buf_name

Changes the active text buffer to a buffer named **buf_name**. If a buffer with that name does not exist, TED creates it.

$

Goto spawned process prompt.

$ command	Execute **command** at spawned process. Returns to TED when command is complete.
W132	Set VT[a] terminal to 132 column width.
W80	Set VT terminal to 80 column width.
STatus	Displays status information on all the text buffers in use.
<HELP>	The HELP key will display help information on all the commands available.

a. VT terminals only.

STatus	Displays status information on all the text buffers in use.
<HELP>	The HELP key will display help information on all the commands available.
PRINTHELP	Prints out list of current help screens for TED. The text file is called TEDHLP.LIS and will be put into the directory the user is currently in.
I STRING	Insert a new line consisting of STRING after the current line. The editor displays this line.
N	Next line (If at EOF, goes to TOF).
R STRING	Replace the current line with STRING.
T	Top of text.
TCR	Shows current Text Catalog Record.
U	Up one line (stops at TOF).
AM	Invokes the ADMINS MANUAL command.
Bo	Bottom (End) of text.
CA	Toggle case sensitivity.
DE [N]	Deletes N lines. The editor displays the line following the last deleted line. If N is absent it is assumed to be 1. If the last line of the file is deleted, the editor beeps as a warning.
A STRING	Append STRING to the current line.
A /STRING/ N	Append STRING to N lines starting at the current line; N is required.
F STRING	Find STRING at beginning of a line.
FN / STRING/ N	Find STRING at beginning of a line N times.
L STRING	Locate STRING anywhere in a line.
LN / STRING/ N	Locate STRING anywhere N times.

C /OLD/ NEW/ [G] [N]	Change OLD into NEW. If G is present, change all occurrences on the line. If N is present, change N lines.
###	(number) A number alone will move to that line number.

J.5.1 Command Line Mode

TED -CLM Command line mode. TED does not go into screen editing mode when this option is used on the command line, and will only accept command line editing commands. See Appendix J.5 "Command Mode (Screen)"

J.6 The TED.ENV File

To change any of the redefinable values used by TED, use the **TED.ENV** file. TED translates the logical name **TED$ENV** to determine the actual file to use as the TED.ENV file.[8] If the logical name TED$ENV is not assigned TED looks for a file named TED.ENV in your SYS$LOGIN directory.

Each line in the TED.ENV file redefines one value, using the format:

```
keyword=[value]
```

Keywords can be the generic names of function keys,[9] or some other TED default value such as default line length. The default values that may be redefined are described below:

page_overlap=n	Overlap pages by **n** lines, i.e. scroll the page when the cursor is n lines from the top or the bottom.
side_scroll=n	Number of columns to scroll sideways for each sideways scroll movement. The default is 20 columns.
max_line=n	Absolute maximum line length allowed. (Limits the size of any ruler.) Cannot be altered during an editing session
line_length	Sets initial line length. Can be altered during editing session by using or changing a ruler.

8. See Appendix J.6.2 "Alternate TED.ENV files".
9. See Appendix J.2 "TED Function Keys".

linesLimit	Limits the size of the file that can be created or updated to the number of lines specified. For example:
	`linesLimit=8`
	will limit the size of the file or internal text field being edited to 8 lines.
	When an action is initiated that would extend the file beyond the specified limit TED will display this message
	`Exceeded max number of lines allowed (8)`
	and sound a warning bell , then ignore any keystokes until "Enter" is typed to acknowledge the message.
	If a file already exceeds the limit when opened by TED, TED will not allow additional characters to be entered into the file.

init_sequence= <sequence>	Send the following escape sequence to the terminal before entering screen mode. A non- printable character may be specified as nn, where 'nn' is the ASCII decimal code for the character (i.e. the <ESC> character may be specified as 27, the SPACE character as 32, and because of the syntax, the '$' character must be specified as 36). For example, to make sure all highlighting is off, put the following in TED.ENV: init_sequence=27[0m
exit_sequence= <sequence>	Send the following escape sequence to the terminal before exiting TED. Use the same syntax as **init_sequence.**
display_xpos	Display the current cursor position on the status line, together with the actual character count. The cursor position and character count are displayed in the form Pos=xxx:ccc, where xxx is the cursor's current column position (the leftmost column position is position 0) and ccc is the character count between the leftmost column and the current cursor position. TAB characters, control sequences etc. can cause the character count and cursor position to be different.
decimal_point=,	Tells TED to use **, (comma)** as decimal point (in rulers etc.). (The expression 'decimal_point=.' is also legal, but has no effect, as it is the default).
save_changes=nn	Tells TED to automatically save the main buffer file when the specified number of changes (i.e. keystrokes that change the file) are made in the main buffer. ('nn' must be between '20' and '4000'). This technique minimizes the chance of losing large amounts of work via system or user error. However, giving save_changes a low value can be inefficient, degrading system performance by causing too-frequent re-writes of large files to disk.
no_write	Disables **WRite filename** function on the command line.
no_include	Disables **INcl filename** and **INRO filename** functions on the command line.

no_execute	Disables **$**(access to spawned process) and **$ command** (execute spawned process) functions on the command line.
no_squeeze	Prevents TED from squeezing out trailing blanks when starting, exiting, or loading text.

If the keyword is a function key, it should have a

```
%(per cent) sign in front of it, e.g.:
%cut=F6
%paste=GOLD+F6
```

which would redefine **CUT** from the default **KP_6** key to the **F6** key, and would redefine **PASTE** to GOLD+F6.

GOLD is used as a prefix to other keys to provide more function keys on the keyboard. If one such prefix key is not enough for the way you want to remap the keyboard, TED provides the ability to define up to three more such prefix keys, named **GREEN, RED** and **BLUE.**

These additional prefix keys are defined in the TED.ENV file by using the **#set** command, e.g.

```
#set BLUE=PF3
```

Once **BLUE** is defined in this manner, any TED function may be defined as **BLUE+key,** e.g.:

```
%status=BLUE+S
```

would redefine the **STATUS** function to be invoked by the **BLUE** key followed by the character S.

Use the #set command also to change the behavior of the TAB key. The following options are implemented:

```
#set TAB=IGNORE
```

to ignore the TAB key.

You can also assign named keystroke functions to the TAB key, where the named functions are a recognized string of characters, e.g. SPACE for a single space and RETURN for an "Enter" keystroke. Single characters separated by spaces represent "typing" that character itself. So use

```
#set TAB=SPACE
```

to change the TAB key into a single space character, and

```
#set TAB=SPACE SPACE SPACE
```

to change the TAB key into three spaces, or use

```
#set TAB=S i n c e r e l y RETURN
```

to change the TAB key into "Sincerely" followed by a carriage return.

Function keys that can be redefined in the TED.ENV file are:

%ascii_table=	View ASCII table
%beg_line=	Cursor to beginning of line
%bold=	Boldface next character(s)
%bot_screen=	Cursor to bottom of screen

%center=	Center text within line
%change_case=	Change case of character(s)
%code_sens=	Toggle code sensitivity
%code_doc=	Document code line
%command=	Enter command mode
%copy=	Copy selected text into paste buffer
%cut=	Cut selected text
%del_char=	Delete character under cursor
%del_endline=	Delete to end of line
%del_line=	Delete line under cursor
%del_word=	Delete word under cursor
%downarrow=	Move cursor down one line
%end_line=	Cursor to end of line
%end_of_file=	Cursor to end of file
%end_word=	Cursor to end of word
%enter_search=	Enter search string
%erase_char=	Erase character in front of cursor
%erase_control=	Erase control line above cursor
%erase_line=	Erase characters on line in front of cursor
%erase_word=	Erase word to the left of the cursor.
%exit=	Exit
%ext_char=	Extended character set
%file_doc=	Insert file documentation
%font=	Invoke font selection window
%frame	Toggle Frame Drawing mode
%gold=	The GOLD character
%goto_line=	Goto line number
%help=	Invoke HELP
%home=	Cursor to first character on line
%indent=	Automatic indent of new line
%insert_line=	Insert line at cursor position
%ins_grid=	Insert grid above cursor line
%ins_heading=	Insert heading above cursor line

%ins_ovr=	Toggle insert/overlay mode
%italic=	Print next character(s) in italic
%leftarrow=	Move cursor left one position
%line_draw=	Toggle Line Drawing mode
%next_line=	Cursor to beginning of next line
%next_para=	Cursor to next paragraph
%next_word=	Cursor to beginning of next word
%normalize=	Normalize text between rulers
%page_marker=	Insert page marker/set page width
%paste=	Past content of paste buffer
%pgdown=	Move cursor down one screen
%pgup=	Move cursor up one screen
%prev_para=	Cursor to beginning of previous paragraph
%prev_word=	Move cursor to beginning of previous word
%printer_ctrl=	Enter printer control screen
%refresh=	Refresh screen (repaint)
%rev_search=	Reverse search order
%rightarrow=	Move cursor right one position
%ruler=	Edit ruler
%scroll_left=	Scroll the text to the left
%scroll_right=	Scroll the text to the right
%search=	Search for occurrence of search string
%select=	Toggle select on/off
%spec_ins=	Insert special character
%status=	Display buffer status information
%swap_char=	Swap characters under cursor
%top_of_file=	Goto top of file
%top_screen=	Cursor to top of screen
%undel_char=	Insert last deleted character
%undel_line=	Insert last deleted line
%undel_word=	Insert last deleted word
%underline=	Underline the next character(s)
%uparrow=	Move cursor up one line

%view_code=	View text and hidden codes

The keyword **#define** allows you to define **macros** in the TED.ENV environment file. Macros allow you to simulate the typing of several keystrokes with one keystroke.

The syntax is:

```
#define func_name=KEYSTROKE sim_key_1 sim_key_2 ...
```

where:

func_name	is a name for the function defined by the macro;
KEYSTROKE	is the key that will invoke the macro function, e.g. F11, CT_B (for Ctrl_B), GOLD+N (for GOLD followed by N), etc.; and sim_key_1 sim_key_2 ... are the sequence of keystrokes that the macro will simulate typing.
sim_key_1 (etc.)	In specifying the sequence, a single character represents that character, more than one character represents physical key names as described below, and any character sequence starting with % represents a function name, either of a TED function key or the function name of a macro previously defined in the TED environment. To define **GOLD M** to mean go to the **MAIN** buffer (e.g. the EXIT keystroke, followed by "=MAIN" and a carriage return at the "Command:" prompt), place the following line in TED.ENV: #define goto_main=GOLD+M %exit = M A I N CR Observe that each character that represents a single character, like all the characters in **=MAIN** must be separated by a space to be recognized as single characters.

The physical keys on the keyboard have been assigned unique names which have to be used when redefining the function keys. These special names are:

VT Keystroke	Workstation Keystroke
PF1 - PF4	F1 - F4
F7 - F14	F5 - F12
HELP	INSERT
DO	HOME
F17 - F20	PREV
FIND	DELETE
INSERT	NEXT
REMOVE	UPAR
SELECT	DOWN
PREV	LEFT
UPAR	KP_0
DOWN	|
LEFT	V
RIGH	KP_9
KP_0	KP_.
|	KP_-
V	KP_,
KP_9	KP_E
KP_.	KP_/
KP_-	
KP_,	
KP_E	

J.6.1 TED.ENV Example

Assuming we have a standard DEC VT terminal keyboard, we might want to create the following TED.ENV file (to emulate the OpenVMS EDT editor):

```
%select=KP_.
%next_word=KP_1
%top_of_file=GOLD+KP_5
%end_of_file=GOLD+KP_4
%command=GOLD+KP_7
line_length=72
#define kpdent=KP_E CR
#define goto_main=GOLD+M
%exit = M A I N CR
```

J.6.2 Alternate TED.ENV files

By default, TED will look in the SYS$LOGIN directory for the TED.ENV file. You may override this by assigning the full directory and file specification of an TED environment file to the logical name TED$ENV. E.g.:

```
$ ASSIGN DISK4:[MYDIR]MYTED.ENV TED$ENV
```

would tell TED to use DISK4:[MYDIR]MYTED.ENV as its environment file instead of the standard SYS$LOGIN:TED.ENV.

J.7 Using Buffers

Provided sufficient memory is available, TED may have up to eight text buffers active at any time (plus the cut_and_paste buffer). Buffers may be created by using the command line command

```
include filename =bufname
```

or by

```
=bufname
```

to a non-existent buffer (in which case you will be asked to verify that you actually want to create the buffer).

The use of buffers can greatly enhance your productivity. Assume that you are making a change to text and need a piece of text from another document. You may then **INclude** the other document into a separate buffer, select the text you need, cut it, then go back to your main buffer and paste it in. When it exits, TED reminds you if you forgot to save any buffer where changes have been made.

J.8 The Text Initialization File

A common need when creating a new piece of text, i.e. when the TX or TI field being edited is empty, is to initialize the field with some predefined lines of text, possibly containing data from TRANS' virtual record at the point the TED editor opens the text field. What follows describes how this initialization can be achieved in TRANS.

TED initializes documents using files called **text initialization files.**[10] Initialization files can be specified as an attribute of a text field in the data dictionary (see Appendix I.4.3 "Text Fields"), or can be controlled via the RMO (see Section 16.24 "TX$INITF: Automatic Initialization of Text Fields"). Literal text, substitutable parameters, and processing logic that will be used to initialize the document are stored in the text initialization file. All lines in the text initialization file will be included at the top of the new document, before the initial ruler of the document. **(the text initialization file should contain its own rulers),** according to the following rules and syntax:

1. Any characters except those characters enclosed in <> (angle brackets) will be inserted as part of the line. Angle brackets identify substitutable parameters, which prompt the user for input and provide information about how the response should be handled, e.g.:

   ```
   <%16sPrompt..:>

   will prompt

   Prompt..: _
   ```

   at the terminal, and accept a string up to 16 characters in length in response. The response is substituted for the string in angle brackets.

   a. The general syntax for creating < > prompts and specifying how to handle the answers is:

      ```
      <%lSppppp>
       ³³³ÀÄÄÄÄÄÄÄÄÄ> Prompt
       ³³ÃÄÄÄÄÄÄÄÄÄ> S=Convert response to uppercase
       ³³ÀÄÄÄÄÄÄÄÄÄ> s=Accept as is.
       ³ÀÄÄÄÄÄÄÄÄÄ> Max. length of answer to prompt
       ÀÄÄÄÄÄÄÄÄÄ> The '%' character must be present.
      ```

   b. If the prompt text ("ppppp" in the above diagram) begins with the two character string **"L$"**[11] TED will assume that the parameter prompt specifies a logical name, and will attempt to satisfy the parameter by translating that logical name, instead of prompting on the terminal. If the indicated **L$** logical name is not assigned, then TED prompts the terminal as usual.

2. If the prompt is enclosed in **<< >> (double angle brackets),** the response is **optional.** The whole line containing the double angle brackets will **disappear** if the response is CR.
3. If the **< > (single angle brackets)** are followed by the **" (ditto)** character the same prompt is repeated until a null response (CR) is given. A new line is inserted into the document for each non-null response.

---

10. AdmIE uses an initialization file when bringing text into internal text fields that have been designated as RTF fields. For more information about this initialization file, refer to Section 17.5.1.1 "Initialization File for Acquiring Internal Text"
11. Parameter behavior can be modified by placing the keyword NOL$PROMPT on a line by itself in the TRANS$ENV file. Refer to Section 6.17.11 "NOL$PROMPT - Don't prompt for L$ parameters (text initialization)" for more information.

4. The following special parameters do not prompt at the terminal, but are automatically loaded by TED if present in the initialization file.

```
<%FILENAME>  For TX fields only.
             The name of the file. This includes
             FILE_NAME.FILE_TYPE, where FILE_TYPE
             cannot be more than three characters.

<%FILE_NAME> For TX fields only.
             File name without the file type, and without
             any directory (path) names.

<%FILE_TYPE> For TX fields only. File type

<%TODAY>     Today's date
```

5. The initialization file may contain conditional statements, as follows:

#ifnew	True if file is empty
#ifold	True if file is not empty
#iftrue < >	True if the < > **prompt** is answered by Y.
#else	True if **#if** is false.
#endif	Terminates an **#if** range. An **#if** range must always be terminated by an **#endif** statement. If/endifs can be nested up to 16 levels deep.

6. The initialization file may have indirect file references similar to what is standard in ADMINS source programs, i.e. it must start in column 1 with "@@" followed by the file name, and be the only thing on the source line, e.g.

```
@@MYDIR:Text1.txt
```

This initialization file should be in RTF format[12].

The indirectly referenced file (in our case MYDIR:Text1.txt) should be a plain text file, but may contain RTF tokens. E.g whenever you want a NewLine (LF) to be inserted, put the "\par" token into the text. E.g. if you want the text in MYDIR:Text1.txt to come out like:

```
This is line 1.

And this is line 3.
```

you should put in:

```
This is line 1.\par
\par
And this is line 3.\par
```

As in any other part of the initialization file, this indirect file reference may contain an ADMINS parameter:

12. Create the RTF file that contains the indirect references in an RTF editor such as MS Word or AdmTed, then edit the file with a non-RTF editor (such as Notepad) and put newlines before and after lines the @@filename indirect references (so that when you are done the references appear on a line by themselves in the RTF source.

```
@@MYDIR:Text<%L_TYPE>.txt
```

where L_TYPE would be a logical name. The value of the logical name is used to build the file specification for the indirect file reference.

A sample initialization file might look like this:

```
Customer: <%40sL$CUSTNAME>
Account.: <%12sL$ACCOUNT>
 Date....: <%TODAY>
```

This initialization file could be used with an RMO that would, for example, load the logical names L$CUSTNAME and L$ACCOUNT with the customer's name and account number from the current customer record. The result would be a file initialized with the following when TED first displays it:

```
Customer: Ms. Jenny Lee
Account.: X92080215588
 Date....: 12-Jan-1991
```

J.8.1 TX$OPTION - Setting Various Options for Internal Text Editing

The reserved field TX$OPTION/I may be used to set various options for internal text editing. The following options are defined:

1	Run "Initfile" always. If the field already contains text, this text is replaced by the output from "Initfile".

J.9 Special Heading and Footing Control Words

As explained in Appendix J.2 "TED Function Keys", the **INS_HEADING** and **INS_FOOTING** keystrokes create two control lines that enclose, respectively, a **Heading** or a **Footing.** Whatever is entered between these two control lines will be printed at the top (Heading) or bottom (Footing) of every page. A sample of a **Heading** might look like this:

```
-------------------- S t a r t   h e a d i n g -----------------
EXAMPLE HEADING                                        Page %PG
Section 1: Demo          Draft %DATE

--------------------- E n d    h e a d i n g -------------------
```

Note that the Heading contains two strings that start with %**.**

The special keywords listed below, all beginning with %**,** indicate positions in the heading (or footing) into which TPR (see Appendix J.12 "Printing Text Fields: TPR") will substitute the specified values at print time.

%PG	Page number
%PG0	Page number initialized. Resets page counter to 1.
%DATE or	Today's date in the form.
%TODAY	**12 - DEC - 1988**
%DAY-DATE	Today's date in the form **Mon 12 - DEC - 1988**

New headings and footings are saved at the point they are found in the text, and are used at the next page break, resulting either from a **PageBreak** control line, or because the page is full. The new footing is used immediately to determine when a page is filled (leaving room to print itself at the bottom). The new heading prints at the top of the next page begun after it is introduced. If you want to change heading information in the middle of your text, make sure you define the new heading before any text is written on the new page.

Note that %**PG** (the page number) will be right justified if the position to the immediate left of the %PG keyword is blank, otherwise it is left justified. Thus, page number 1 for

Page %PG

will print:

```
Page   1
```

while

Page A-%PG

will print:

```
Page A-1
```

J.10 Including Font Codes with TED

Text fields may include Text Font specification using the same ADM$STYLE mechanism as RNF and REPORT.

First, the TED.ENV file (see Appendix J.6 "The TED.ENV File") must set mnemonics for the fonts that you want, using the "font=" keyword:

```
font=Courier_12
```

The mnemonics must correspond to names used on

```
.CC MNEMONIC control_sequence
```

lines in the file assigned to the logical name ADM$STYLE at print time.

Up to 16 different "font=" lines can be present in the TED.ENV file.

In TED, to activate a new font press the **font** key[13] **(CT_F,** by default). A pop-up window displays the available fonts. Move the cursor to the font you want and press SELECT.

(You cannot see the text in the selected fonts on the terminal, but it will appear in the printed output.) However, the name of the font in effect does appear on the message line.

When TPR (see Appendix J.12 "Printing Text Fields: TPR") prints the text it inserts the actual control sequence for the fonts selected at the points you have specified. TPR finds the appropriate control sequence by looking up the mnemonic in the file assigned to the logical name ADM$STYLE.

If the mnemonic is not found by TPR in the "ADM$STYLE" file, the font currently in effect is used.

13. This key may be redefined in the TED.ENV file by placing the **%font=KEYNAME** line in the TED.ENV file.)

J.11 The Printer Control Screen

Printer setup characteristics can be set for the document using the **Printer Control** screen, invoked by the **PRINTER_CTRL** keystroke. The **Printer Control** screen contains the following information:

```
*-----------------------------------------------------------*
|                ADM$TI_nnnnnnnn Printer Control            |
|                                                           |
| Characters per inch.....:  10                             |
| Lines per inch..........:   6                             |
| Print lines per page....:  60                             |
| Page size in lines......:  66                             |
| Automatic indent columns:   5                             |
| Orientation (0, 1 or 2).:   0   1=Portrait, 2=Landscape   |
| Starting page number....:   0                             |
|                                                           |
| Printer initialization..: 00 00 00 00 00 00 00 00 00 00   |
|                                                           |
| Printer termination.....: 00 00 00 00 00 00 00 00 00 00   |
 | CT_Z to return to text editing
  *---------------------------------------------------------*
```

When TPR is given the document to print, it outputs no control sequence for attributes that are set to zero. **Printer initialization** allows you to enter hexadecimal ASCII codes (e.g. the escape character is 1B) for up to 10 characters to send to the printer before the document is printed. This could be used to send setup control sequences if TPR does not have the proper setup for your printer built in. **Printer termination** can be used in a similar way to reset the printer after the document is completed.

If **Orientation** is **0,** TPR sends no orientation control sequence to the printer. If **Orientation** is set to **2** (landscape), TPR will send the **landscape** control sequence command to the printer. After the document is finished printing TPR will reset the printer to portrait.

J.12 Printing Text Fields: TPR

Documents stored in ADMINS internal text fields can be printed using the TPR[14] text printing utility. TPR's syntax for ADMINS internal text fields is:

```
$ TPR -INT -FIELD=fieldname -KEY1=value file.mas
```

If the file has more than one key, supply additional key values by adding the keywords:

```
-KEY2=value -KEY3=value ...
```

If all the arguments are not given on the command line TPR will prompt for them (assuming TPR special logical names are not assigned, as explained below):

```
$ tpr -int
File name: journal.mas
Enter value for key field MSG: 910822
Enter Text Field Name to print: comments
```

TPR -INT will translate the following logical names to satisfy its arguments:

```
TPR$FILENAME

TPR$FIELD

TPR$KEY1 TPR$KEY2 ... TPR$KEYn
```

$ TPR -INT (with no other command line arguments) will **first** try to translate these logical names. It will only prompt for a value if the corresponding logical name for that value is not assigned. The following logical name assignments could replace the TPR -INT dialogue in the example above:

```
$ASSIGN JOURNAL.MAS TPR$FILENAME

$ASSIGN 910822 TPR$KEY1

$ASSIGN COMMENTS TPR$FIELD
```

One easy way to print internal text using TPR from an RMO running with a TRANS screen is to set up a symbol as follows:

```
TPRINT :== "$ADM$DIST:TPR -INT"
```

Then set the TPR arguments into the special logical names described above (use the CRLOG subroutine), and call the SPAWN subroutine to execute TPRINT.

When printing internal text with TPR, both the HEADING and PAGEBREAK functions of TED will work. To avoid having TPR put out a default heading if no heading is provided in the text, add the -NOP keyword on the command line, e.g.

```
TPRINT :== "$ADM$DIST:TPR -INT -NOP"
```

14. TPR will print any text file, but it has the special capability to handle formatting information provided in TED documents.

TPR may also be instructed to print only selected pages of the document stored in the internal text field by using the keywords:

-FROM_page=m

-TO_page=n

on the command line. These page number arguments may also be provided via special TPR logical names:

TPR$FROM_PAGE

TPR$TO_PAGE

TPR command line options are:

-ID	Display TPR version id, and exit
-3LPI	Print double spaced, i.e. 3 lines per inch.
-4LPI	4 lines per inch.
-NOHeading	Do not print automatic file heading. An automatic file heading is printed by default unless file has its own heading.
-NOPage	Do not print automatic heading or page numbers.
-CONsole	Print output to the console (terminal).
-MORe	If output to -CON, display one screen at a time and wait for a keystroke to continue with the next screen.
-TPR	Print at the printer attached to the terminal.
-LANDscape	Print document in Landscape Mode.
-IBM_ Proprinter	Printer is IBM ProPrinter.
-LN03	Printer is DEC LN03.
-HP_Laserjet	Printer is HP LaserJet.
-CI_3500	Printer is C.Itoh 3500
-XEROX	Printer is Xerox.
-IBM	Put HP LaserJet in IBM PC mode.
-FF	Formfeed before first page.
-INDENT=n	Indent **n** characters before printing text.
-CPI=n	Print **n** characters per inch.
-PAGE_ LENGTH=n	Print **n** lines per page.
-COPIES=n	Print **n** copies of text or document.
"=filename"	Output to **filename**, do not spool the output.
?	Invoke TPR Help.

-FORM=form_name or FORM is either the queued printer form name or form

-FORM=form_number number. On OpenVMS systems, FORM is case insensitive.

J.12.1 The TPR.ENV Environment File

To change the default setups for TPR, create a **TPR.ENV** file. By default, TPR will look in the SYS$LOGIN directory for a file named TPR.ENV. You may use another file as the TPR environment file by assigning its full directory and file specification to the logical name TPR$ENV, e.g.:

```
$ ASSIGN DISK4:[MYDIR]MYTPR.ENV TPR$ENV
```

The TPR.ENV file may contain the following entries:

%printer=	Give the name of your printer if it is not an LN03 on OpenVMS, or an HP on UNIX. Allowable values are: IBM_PROPRINTER DEC_LN03 HP_LASERJET CI_3500 XEROX_4045 CONSole If you do not have a printer attached to your PC, use %printer=CONSOLE to direct the output to your screen if you want to view your output using TPR.
%more=YES	Console output will display one screen at a time and wait for a keystroke to continue with the next screen.
%page_length=	number of lines to print per page. For continuous output (i.e. no page breaks), use %page_length=0.
%indent=	number of columns to indent on the page. By default, TPR will indent 5 columns.
%cpi=	number of characters per inch. TPR defaults to 10 cpi. If TPR changes the cpi setting it will reset the power-up settings after printing.
%ff	Send a form feed before printing.
%end_ff	Send a form feed after printing.
%init_sequence=	Send the indicated sequence of characters to the printer before printing the document. Init_sequence is useful for sending printer setup control sequences. Use the character's ASCII decimal code surrounded by dollar signs for non-printing characters, spaces, and the dollar sign, e.g. 27 for the escape character, 32 for the space character, and 36 for "$".
%exit_sequence=	Send the indicated sequence of characters to the printer after printing the document.

%map:ttt=ppp	Used to map internal TED 8-bit codes to corresponding codes for the current printer. (Note that if a %**printer=** statement appears in the TPR.ENV file, it must appear before any %**map:** statements.) **ttt** is the (decimal) three digit position in the ASCII table for the TED character code, and **ppp** is the corresponding three digit position of that character for the printer in use.

The following keywords are used to re-define printer control sequences when output is destined for a printer for which TPR does not provide built-in support. The syntax described above for %**init_sequence** is used to give the decimal ASCII code to be substituted into the TPR output whenever the indicated control sequences appear in the document.

%font_file=	Name of font file to be used. If %**font_file** is not present, the value assigned to the logical name ADM$STYLE is used.
%bold_on=	Control sequence to turn **bold** on.
%bold_off=	Control sequence to turn **bold** off.
%underline_on=	Control sequence to turn **underline** on.
%underline_off=	Control sequence to turn **underline** off.
%italic_on=	Control sequence to turn **italics** on.
%italic_off=	Control sequence to turn **italics** off.
%advance=	Control sequence to perform a 1/2 line feed.
%nohead	*Suppresses the printing of an automatic heading.*
%portrait=	Control sequence to turn on portrait mode.
%landscape=	Control sequence to turn on landscape mode.
%ascii	Print only ASCII characters (throw away escape sequences like bold, underline etc.).
%reset=	Control sequence to reset printer.
%ADM$SPOOLn=	Send output to device assigned to logical name ADM$SPOOLn.

Any line in the TPR.ENV file not starting with the '%' character is treated as a comment line.

Appendix K:Using Text Fields

Documents can be integrated directly into ADMINS data files by using the **TInn** and **TXnn** text field data types. Text data can be displayed, processed, and output using the same ADMINS tools and syntax as any other ADMINS data type, subject to the special conditions set out in what follows.

K.1 Special Considerations

Several special rules, syntax, limitations, conditions, and requirements apply when text field data types are used.

1. The ADMINS Data Dictionary (ADD) **must** be in use, and **text fields must reference dictionary data elements,** either directly (i.e. by defining the file via the ADD screens), or indirectly (via the "@" reference syntax in the ".DEF" file, as described in Section 2.4.2.2 "Referencing Data Dictionary Data Elements").
2. When a file that has text fields is defined three files are created:

```
$ def txtest
DEFSZ: 60 NF: 5  KEYLEN:3  RECSZ: 77  NRECS: 100
 # OF BLOCKS   DATA: 22  INDEX: 3  TOTAL: 25
MU$DUA0:[BD.NEWMSG]TXTEST.MAS;1 CREATED
INDEXED file. KEYS are: MSG
MU$DUA0:[BD.NEWMSG]TXTEST.TCF created
Catalog record size: 148. Catalog header size: 888
MU$DUA0:[BD.NEWMSG]TXTEST.TSF created
Storage record size: 511. Storage header size: 511
```

   The two additional files that are created support the text field data types:

   - **TCF -** The **Text Catalog File** stores information that relates the records in the ADMINS data file to the records in the Text Storage File.
   - **TSF -** The **Text Storage File** actually stores the text itself in the case of TInn ("internal text") fields, or stores the **name** of a text file in the case of TXnn ("external text") fields.

These two files, together with the ADMINS data file, are treated as a single entity by ADMINS. But, because three different files exist as far as the file system is concerned, **special care must be taken when system operations (DCL, RMS, etc.) are used to handle ADMINS data files that contain text fields.** For example, if any of the following DCL commands, or any commands that perform similar functions, are used to process an ADMINS data file that contains text fields, you must **make sure** that the ".TCF" and ".TSF" file are also appropriately processed.

RENAME COPY DELETE PURGE BACKUP

Take the following command:

```
$ COPY TXTEST.MAS NEWTX.MAS
```

The DCL COPY command would create a new data file, NEWTX.MAS. But an attempt to use NEWTX.MAS in ADMINS would fail because the file contains text fields and ADMINS would not find the TCF and TSF files.

```
$ tra newtx.mas
Open failure on MU$DUA0:[BD.NEWMSG]NEWTX.TCF
%NONAME-F-NOMSG, Message number 00000004
```

To correctly copy TXTEST.MAS to NEWTX.MAS use:

```
$ COPY TXTEST.MAS,.TCF,.TSF NEWTX.*
```

Users must have read/write access to TCF and TSF files. Any open of an ADMINS data file that contains text fields, even if it for read-only access, will fail unless the TCF and TSF files can be opened for writing.[1]

3. TCF and TSF files are needed to support text fields. Consequently, **text fields must always be an actual field** in an **existing record** of a data file. Some examples of situations where text fields cannot be used, because of the failure to meet this requirement:

   "Pure" **local RMO fields** that do not map to, for example, linked fields in a TRO, cannot be text fields. (Reason: no actual field in data file.)

   In **Append Mode in TRANS** you cannot view or edit the text fields of the main file. (Reason: the record is not appended to the file until you press NEXT, so there is no "existing record" that can be tied to the TCF and TSF files).[2]

   A **CREATE** statement in REPORT or the ANALYZER, or a **VIRTUAL** statement in SCREEN cannot be used to create a text field. (Reason: no actual field in data file.)

4. Text fields **should not** be assigned a value in an expression. For example, the following expressions:

   ```
   TX_FLD = ' '
   TI_FLD1 = TI_FLD2
   ```

   will not work, because the value is changed only in the ADMINS data file, not in the Text Storage File. To copy text from one text field to another, use the TEXTCOPY[3] subroutine.

---

1. TCF and TSF files can be dynamically expanded by multiple users. OpenVMS restricts access to files with this capability, such that only users with write access can share the file.
2. In **Insert Mode** of TRANS you may view or edit text fields, because the record is created as soon as you "Enter I to Insert"
3. See Appendix H.6.1 “TEXTCOPY: Move Information Between Text Fields”.

5. When **external text fields (TXnn) are used,** the logical name

   ADM$TX_DIRECTORIES

   must be assigned. ADM$TX_DIRECTORIES must point to a text file which contains a list,[4] one entry per line, of logical names for all directories that are to be used for storing and searching external text files.

   The logical names in the ADM$TX_DIRECTORIES list are all translated whenever ADMINS opens a data file that has external text fields. When an external text file is referenced, ADMINS will search for the completely expanded disk and directory specification (i.e. all logical names translated, all default values included) of that file among the **translations** of the ADM$TX_DIRECTORIES logicals. If a match is found, ADMINS will use the **corresponding logical name** for that disk/directory in building its internal file specification for the referenced file that will be stored in the text storage file. This is done so that changes in the physical location or organization of the text files will not affect their use with external text fields.

   If no match is found for the expanded file specification, ADMINS displays the message:

   ```
   <Directory> is not a valid Text Directory
   ```

   To make the invalid directory valid, assign its specification to a logical name, and include that logical name in the ADM$TX_DIRECTORIES file.

   For example, say the text files associated with our data file are stored in two directories on disk T1$DISK, [LETTERS] and [MEMOS]. If the following logical name assignments are made:

   ```
   $ ASSIGN TI$DISK:[LETTERS] LET$DIR
   $ ASSIGN TI$DISK:[MEMOS] MEM$DIR
   ```

   We would then make the following entries in the file assigned to the logical name ADM$TX_DIRECTORIES:

   ```
   LET$DIR
   MEM$DIR
   ```

   Then, when TRANS is used to edit the (previously empty) external text field TX_TEST, it will prompt for the name of the external file to be associated with field TX_TEST. If we give the following file specification (assuming our current default disk/directory is TI$DISK:[WORK])

   ```
   Enter file name: [MEMOS]91VAC.TXT
   ```

4. If an asterisk appears in column one in the ADM$TX_DIRECTORIES file that line is treated as a comment line. Any text in the line after the logical name followed by at least one white space character is also treated as a comment.

ADMINS first fully expands the disk/directory portion of the file specification. In this case it would be expanded to:

```
TI$DISK:[MEMOS]
```

Then ADMINS searches for this disk/directory in its list of translations of the logical names in the ADM$TX_DIRECTORIES file. In our example, that list can be represented by the following table:

```
ADM$TX_DIRECTORIES
      Entry              Translation
------------------      -----------
    LET$DIR             TI$DISK:[LETTERS]
----> MEM$DIR      ----> TI$DISK:[MEMOS]
```

ADMINS would find a match for the specification given in the translation for the logical name MEM$DIR, so it would store the following as the complete file specification in the Text Storage File:

```
MEM$DIR:91VAC.TXT
```

This method allows movement of the text files associated with external text fields to different physical locations, while only having to account for the movement once, **by changing the logical name assignment.** Continuing the example, if it became necessary to change the location of the [MEMOS] directory to disk UT$DISK, the logical name assignment:

```
$ ASSIGN UT$DISK:[MEMOS] MEM$DIR
```

is the only change necessary.

The **ADM$TX_DEFAULT** logical name is optional. If assigned, and if present in the ADM$TX_DIRECTORIES file, ADMINS will use it as the default location where external text files are to be created or searched for. If ADM$TX_DEFAULT is not assigned, the current default directory will is used.

6. Whenever a text field is included in a file definition, ADMINS **automatically creates an internal field** (field type L) called **TI$fieldname.** For example, if the internal text field EXPLANATION/TI60 is included in a file's definition, when that file is defined it will actually have the following two fields included:

   ```
   EXPLANATION TI60
   TI$EXPLANATION L
   ```

   These special internal fields relate the record in the ADMINS data file to the appropriate record in the text catalog file (TCF) (which in turn relates the data file record to the text storage file (TSF) record).

   The TI$ fields are real fields in the file, and count against the ADMINS limits such as the maximum number of fields in a file or virtual record. But these fields are **always handled internally by ADMINS** and **should never be manipulated in any way** (i.e. edited, changed, linked, transferred etc.), and need never be referenced by application code or by the user.

7. Text fields cannot be key fields.
8. Text fields should not be used as arguments for ADMINS subroutines unless the documentation for the subroutine specifically states that text fields are supported (see Appendix H: “Subroutines”).

9. When the TED edit of the document associated with a text field is concluded, and the new or changed document is saved, **the record that contains that text field is written back to disk immediately.** This differs from standard TRANS functionality, and in certain circumstances the difference may be significant.

   For example, if the text field is a LINKed field, the LINK record buffer is written back immediately and unconditionally. (Normally, LINKs are not written until end-of-record processing, or LINK writing can be controlled via the special RMO field W$W (refer to 16.1"Controlling Changes Written To Disk").

   Application developers must take this immediate writing into account when a text field can be altered in a screen. Function and controls such as those listed below may be seriously affected by the use of text fields that can be altered in a screen.

   W$W to control writing to main or link files, (refer to 16.1"Controlling Changes Written To Disk").

   NOWRITE inhibits field by field writing of main file (refer to 16.1.1"High Volume Update: NOWRITE").

   LFEXIT or LFBACK control of data entry (refer to 6.3.1.1"Update Mode Under LFEXIT Control").

   REQUIRE statements to ensure field is non-null before it is written (refer to 5.5.5"REQUIRE Statement").

10. Text fields cannot be part of a Data View (Appendix I.9 "Data Views ").

Appendix L:Managing ADMINS

This Appendix describes ADMINS system management issues. Detailed knowledge of these items is not needed for ADMINS application development.

L.1 Version and Date Stamp of an ADMINS Command

You can determine the version and date (i.e. the date the image was created) of any ADMINS command by typing "command -ID". For example:

```
>adman -id
AN          ADMINS V8.4 Win32 3-Mar-2006
            ADMINS File System
LIBADMINS:  ADMINS V8.4 Win32 3-Mar-2006
LIBADMIO:   ADMINS V8.4 Win32 3-Mar-2006
LIBSUBS:    ADMINS V8.4 Win32 3-Mar-2006
LIBVTIO:    Dummy
LIBLOG:     ADMINS V8.4 Win32 3-Mar-2006
```

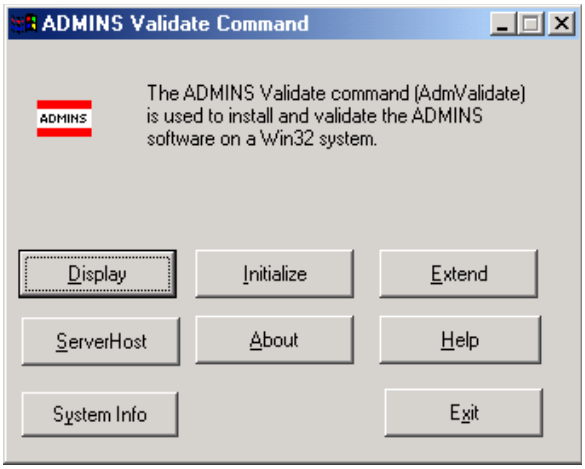
Use the version and date stamp of an ADMINS command in conjunction with the ADMINS Release Notes Addenda that are sent out periodically to determine if the problems and fixes described apply to the images currently in use on your system.

When you contact ADMINS Technical Support with a question or problem related to an ADMINS command currently in use on your system, please provide the version and date information for the image.

L.2 ADMVALIDATE

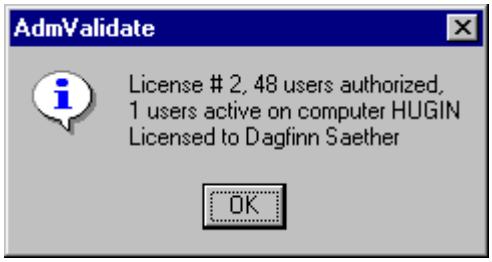
A typical ADMINS installation (see the ADMINS Installation Cookbook) for more information about installation) consists of a server computer running the ADMINS Lock Manager service, and a number of client computers running the ADMINS client software (e.g. TRANS, MAINT etc).

The ADMINS AdmValidate command is used to permanently install the ADMINS software on the server. If an ADMINS installation is not verified by running the AdmValidate command, all ADMINS images will time out 3 months after their build date

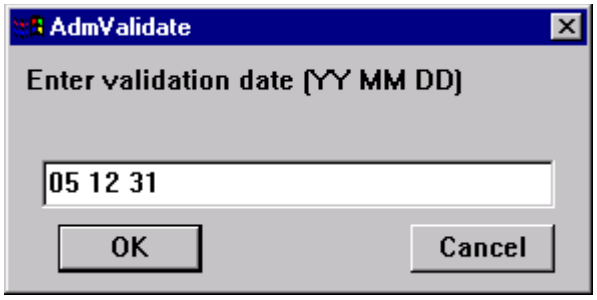


Before AdmValidate is run, a normal ADMINS Installation Procedure has to be run, and the ADMINS Win32 Lock Manager service should be running on the server.

The Display button will show the current license information:

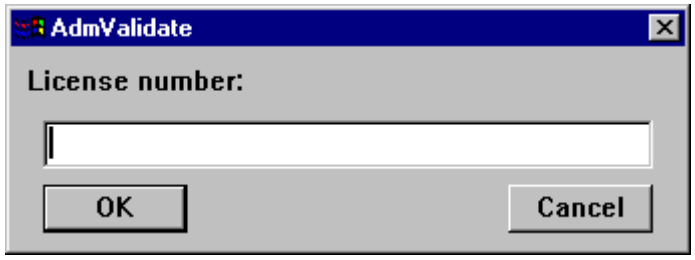


To install the software, or to extend the termination date of an installation, use the Initialize button. You will be prompted for the following information:

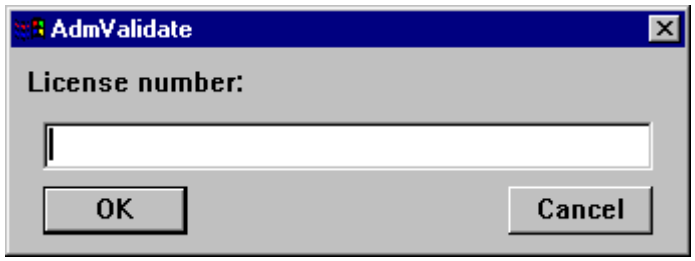


A Validation Date is the date when the software will cease to operate. Always enter only two digits for year, e.g. 05 12 31 for December 31, 2005. If it is a trial installation, or a temporary installation awaiting a permanent license number, use an agreed upon date, e.g. 3 months into the future.

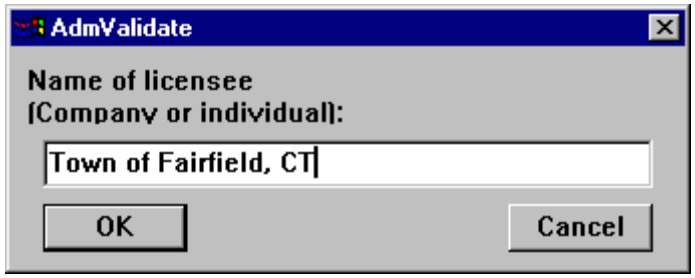
Next you will be prompted for a license number:



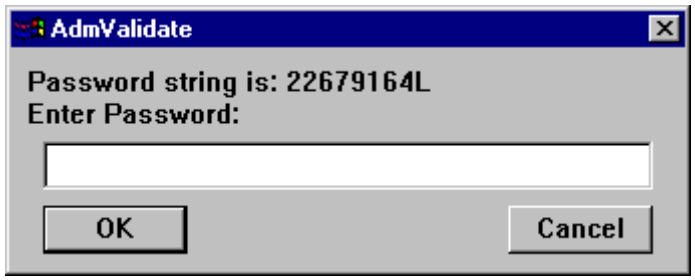
Enter your license number for a permanent installation, or the number of a valid temporary license with the allowed number of concurrent users for a trial or temporary installation.



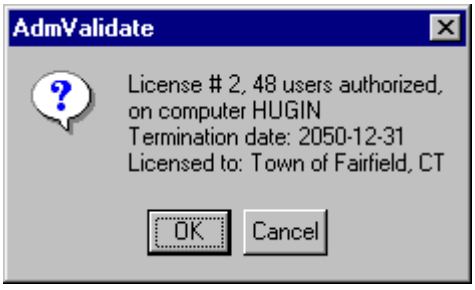
Enter the name of your organization (or your own name, if it is a personal installation).



At this point you will be prompted for a password and you need to be in contact with ADMINS, Inc. to obtain a password. Enter the password exactly as given to by the ADMINS, Inc. representative.

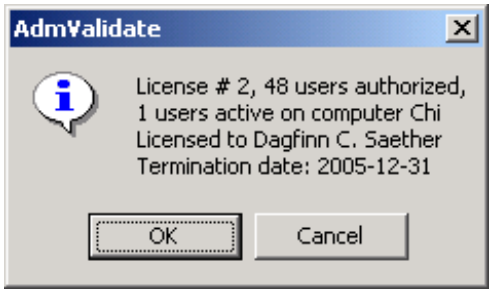


The system will now ask you to verify the license information:

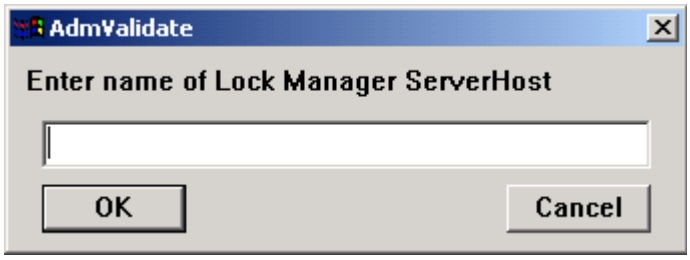


Click on OK (or press Enter) if the information is OK. AdmValidate will now register your installation as being valid till the termination date shown.

The Extend button is used to extend the termination date of a trial or temporary installation once, to give you time to get in contact with ADMINS, Inc. to reach a permanent agreement.



On the client computers AdmValidate is used to identify which computer is running the Lock manager service. Click on the ServerHost button:



Enter the computer name of the computer where the ADMINS Lock manager is running.

L.3 Managing ADMINS Usage Slots

The ADMINS Usage Management File allows a system manager to reserve, or to limit, the number of slots available for a user (that is, a Windows username), or for all users. The Usage Management File can be created with any text editor. An example showing syntax for the various options is shown below.

```
! ADMINS Usage Management File
! ==============================
! Anything following exclamation point is a comment
! Slots reserved for users
! ------------------------
user_name=(public,1)             ! 1 Slot reserved for user "public"
user_name=(tclerk,0,5)         ! tclerk limited to 5 slots
 user_name=(cserv,1,2)         ! 1 slot reserved, limited to 2
 user_name=(  *  ,0,1)         ! all others limited to 1 usage
!             |   | |___________ maximum concurrent logins for user
!             |   |_____________ number of usages reserved for user
!             |_________________ login name
!
```

In this configuration one user logged in as PUBLIC and one user logged in as CSERV are guaranteed access to an ADMINS usage slot at all times. Other users compete for the remaining slots for which the system is licensed. However, the number of concurrent usage slots available to users logged in as TCLERK is limited to five, and two concurrent slots are the limit for users logged in as CSERV.

To enable the Usage Management File, first create the file with the desired options. It can be in any folder. The usage file may have any name. Protection on the usage file must be set so that everyone can read it Then make the following system logical name assignment :

```
admlcr -ts adm_usage_slot <path to usage file>
```

When ADMINS runs, it tries to translate ADM$USAGE_SLOT. If the logical name exists, ADMINS tries to open the file it points to. If for any reason the file cannot be opened or cannot be read (no such file, privilege violation, etc.) ADMINS simply continues. Otherwise, the options in the file are read and used by the installation control system.

Appendix M:The AppMenu SubSystem

The ***Application Menu*** facility ("AppMenu") provides a way to define menu bar items outside of the TRS that appear in every screen that asks for them. The TRANS_ENV file identifies the files to use to construct the menu. In the TRS, the keyword APPMENU (appearing anywhere before the SCREEN keyword) tells TRANS to load the menus specified in the TRANS_ENV file.

The ***AppMenu Parameter Verification*** subsystem ("APV") provides a mechanism to enter, verify and pass parameters to command files and reports being run from the AppMenu system.

The Application Menu facility uses four or five ADMINS files, specified in the TRANS_ENV file using the following keywords:

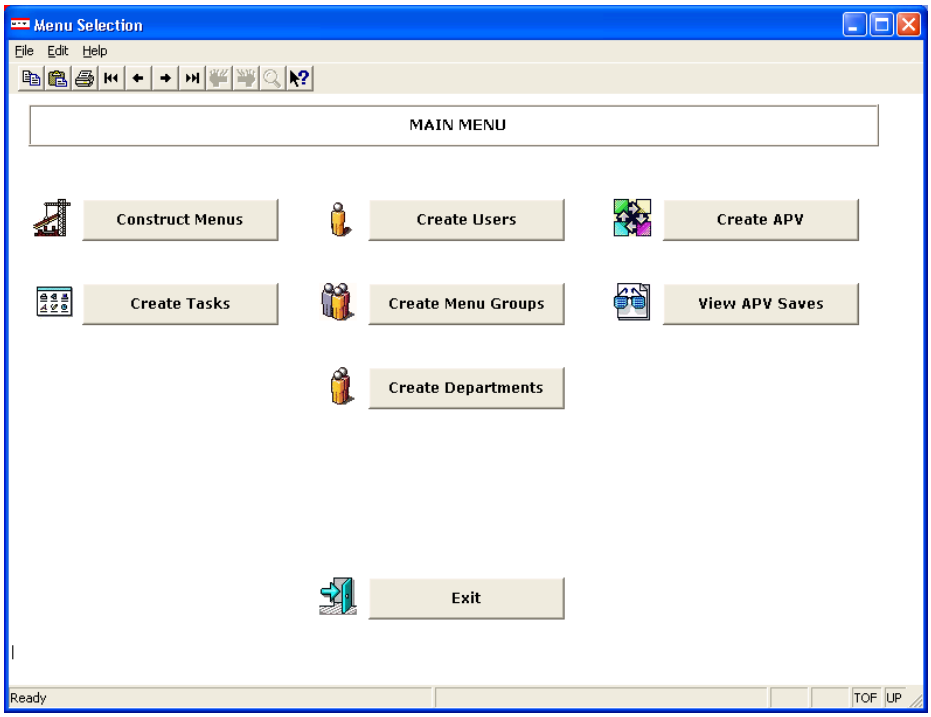
keyword	use
appmenu.userprofile=	name of user profile file
appmenu.menuspec=	name of menu specification file
appmenu.taskfile=	name of task specification file
appmenu.parameter=	name of parameter information file
appmenu.defaults=	name of parameter defaults file (optional)

If necessary, you can maintain a set of files that meet AppMenu requirements by developing your own application. See Appendix M.2 "AppMenu File Specifications" for a detailed description of the required file structures.

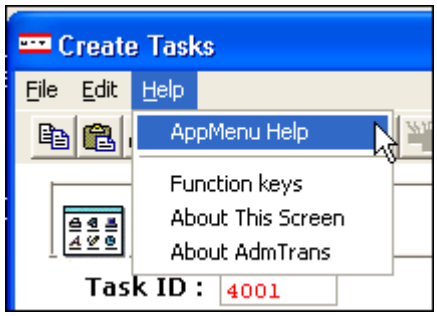
However, most ADMINS developers will find it easier to use the AppMenu Maintenance Utility, an easy-to-use, self-documenting, AdmTrans-based application that includes ready-made file structures that support all AppMenu functionality.

M.1 AppMenu Maintenance Utility

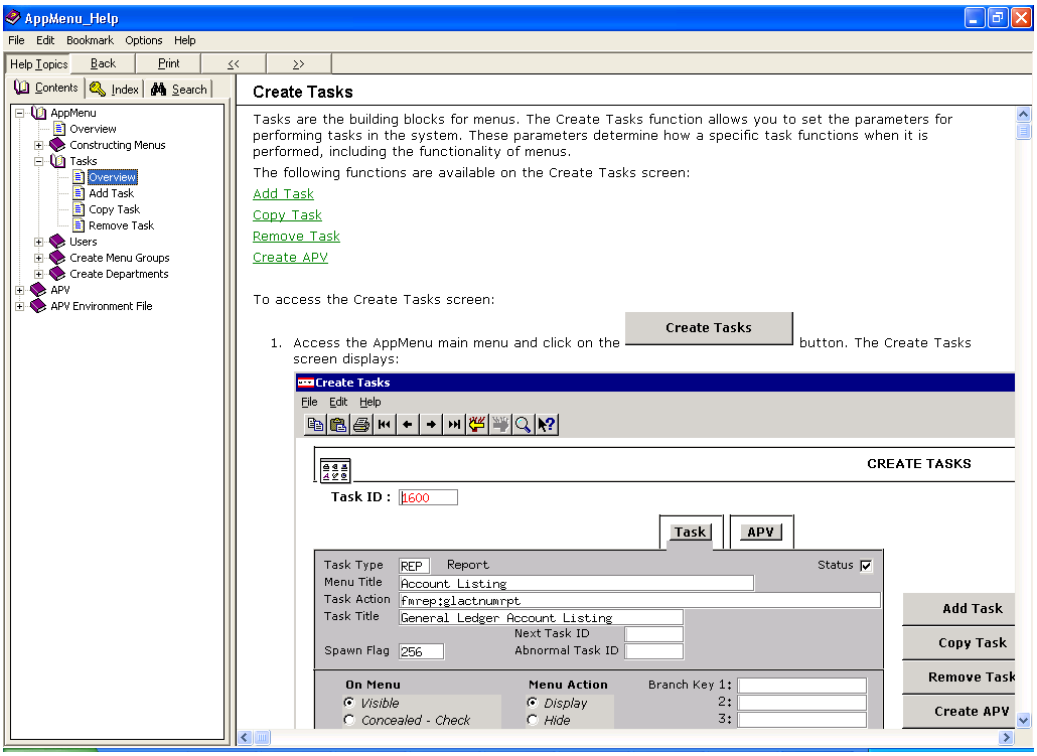
The AppMenu Maintenance Utility is provided in kit form. Obtain it from ADMINS, Inc. and install it and run it according to the instructions provided with the kit.



The various functions available in the AppMenu are completely described in the Help provided. In any of the function screens click the AppMenu Help item on the Help Menu.



to display help for the AppMenu Maintenance Utility using the "Windows Help" facility:



M.2 AppMenu File Specifications

This section contains detailed information about the structure and contents of the five files that are used to specifiy AppMenu. Most developers should be able to specify their menus completely using the AppMenu Maintenance Utility, described in the previous section, and will need to refer to this section only for special circumstances.

M.2.1 User Profile file

The TRANS_ENV file entry ***appmenu.userprofille=FILENAME*** must identify an ADMINS file with the following fields:

username/An	The key field of the file must be an An field containing the user's logon user name in upper case. It must be A8-A24.

MNUGRP/An	If the field MNUGRP with the same type and length as the username field exists, and the field is non-blank, this value will be used to determine which menu to use. This allows menus for several users (e.g. users within a department) that share the same menu item to be specified once.
DEPTCODE/An	Optional field with the same data type as username. If present it may be used to store default values to APV logical names at the department level (see **SAVE** keyword under **APV** files).

M.2.2 Menu Specification file

The TRANS_ENV file entry ***appmenu.menuspec=FILENAME*** must identify an ADMINS file with the following fields:

username/An KEY1	Same data type and length as the corresponding field in the *userprofile* file.
mnulvl1/I KEY2 mnulvl2/I KEY3 mnulvl3/I KEY4	These three integer fields determine the order and structure of the items to appear in the menu. The structure is: i 0 0 Item appears at the top level menu bar. i j 0 Item appears at the first level drop down menu. i j k Item appears at the second level drop down menu.
TSKID/I	An I field containing the key value into the taskfile file, that contains the details about the menu item.
DABTSK/A2	Optional field that may be used to disable a task at the user (or menu group) level. It can be set to blank, 'H' or 'G', see the DABTSK field under the taskfile entry. In order to use this feature when executing a task from the RMO the menuspec file must have an alternate index 1 keyed by: Username/an KEY1 TSKID/I KEY2 Possible additional key fields If this index exists, and the DABTSK field is non-blank the task is disabled if trying to run it from the RMO. If a task is a branch to another screen, the DABTSK field may also be set to the value 'R' to make the target screen Read Only. Thus a screen that is full function for some users may be a Read Only screen for other users. (This is equivalent to having ADM$READONLY assigned for the target screen.)

M.2.3 Tasks file

The TRANS_ENV file entry ***appmenu.taskfile=FILENAME*** must identify an ADMINS file with the following fields:

TSKID/I KEY 1	An I field - identifies each task specified for the menu[a].
TSKTYP/A	Identifies the action to be taken if this menu item is selected. Possible values are:
MENU	A Menu item. Must have next level items.
SCR	A branch to another screen
REP	An ADMINS Report
COM	An ADMINS command file
PERL	A Perl script to execute
RUN	A Windows script (e.g. .BAT or .CMD)
EXEC	Execute RMO with M$M set to value of TSKACT.
ARG	An ADMINS Report Generator spec. The field TSKBKEY1 may contain the type of report to generate (supported types are 'CSV' and 'XML').
HLP	Invoke ADMINS Help
WHLP	Invoke WinHelp
SEP	A separator in the menu list
TSKDESC/An	An An field that contains the text to display for the item.

TSKACT/An	An An field containing the target screen, command file, report, or whatever the TSKTYP requires. E.g.:
	`MY_APP:MYTro/screen`
	`MY_APP:MYComfile`
	For TSKTYPs HLP and WHLP this field must contain the pathname of the help file to use, e.g.
	`MY_APPHELP:Purchase Order.hlp`
	If the TSKTYP is HLP (ADMINS help) a field TSKTIT/An may contain the topic name to look for in the help file. If no topic is provided the name of the current screen will be used.
	For TSKTYP EXEC it must contain the 2 character code to be passed in the M$M field at the resulting RMO call.

TSKBKEYn/An	Multiple fields. If TSKTYP is 'SCR' may contain fields that hold key values for the branch. If TSKTYP is 'REP' TSKBKEY1 may contain output type to generate ('CSV' or 'XML')
DABTSK/A2	Optional field. Set to 'H' to hide a task (it will not appear in the menu even if referenced in the menuspec file) or 'G' to gray it in the menu (will appear in the menu, but cannot be selected for execution).
ERRFLG/A2	Optional field. The procedure associated with this task may set this field to non-blank to signal an error on last run. This task cannot be started from the AppMenu until this flag is cleared.

INUFLG/A2	Optional field. AppMenu set this field to 'Y' to signal the procedure is in use before starting 'REP' and 'COM' tasks. It will be cleared when the task finishes.
SETINU/A2	An optional field that, if present, controls whether the INUFLG flag is automatically set by AppMenu or not. If this field is present it must be set to 'Y' for AppMenu to set the INIFLG field.
MNUFLG/A2	Optional field. If it exists it must be set to 'Y' for a task to be allowed to appear in the menu. If a task is started from the RMO (not be selecting a menu item), AppMenu normally checks if the task is in the users menuspec file (to verify that the user is allowed to run the task). If this field is 'N', this check is not performed.
SPAWNFLG/I	Optional field. May be used to modify the behavior of "spawned" tasks. Valid values are: 256: Run minimized 512: Run in IDLE_PRIORITY_CLASS
NXTSKID/I	Optional field. May contain the task id of the next task to run if the task completes successfully. If the command file, for some reason other than bombing, wants to prevent execution of the NXTSKID task, it may assign a value to the logical name AM_NXTSKID_STOP_nnnn, where nnnn is the task id of the command file being run. If this logical name is assigned when the command file exits the logical name will be deleted, and the NXTSKID task will not be run.
NXTSKID/I	Optional field. May contain the task id of the next task to run if the task completes successfully.
ANXTSKID/I	Optional field. May contain the task id of the next task to run if the task completes unsuccessfully (ERRFLG is set).
LSTUSR/An LSTDAT/DT LSTTIM/TM	Optional fields. If present these fields are filled in with the user, date, and time when this task was last selected.

a. The AppMenu task table is limited to 8192 items.

M.2.3.1 Additional Task file fields (for APV support)

To support the APV subsystem the taskfile has must contain the following additional fields:

PVFLAG/A2	This field must be non-blank to invoke APV subsystem when the menu item is selected.
1MSG/An 2MSG/An 3MSG/An 4MSG/An	Contains a description of the program to be run. Up to four fields may be used to provide the description, but only 1MSG must be present.

RPTTYP/I	A flag to indicate which report types are contained in the .REP (i.e. how the report may run). Valid codes are: 1 Regular report (To preview in a browser or print directly) 2 XML 4 CSV 8 Excel or any combination thereof (e.g. 15 means support for all types). If more than one report type is supported (e.g. 1+2+8 = 11) by default the lowest supported number (in this case (1=Regular report) will be the default report type to run. If you want one of the other report types to be the default, add TYPE * 1000 to the report type (e.g. if in this example you want the XML report to be the default, specify RPTTYP = 2011). A ***Regular report*** means that a traditional ADMINS report will be produced, which either can be directly spooled to a printer queue for printing, or it may be previewed in a browser (the user has the option to print it from the browser). If the Preview option is chosen the output from REPORT will be previewed using TedRE. If you do not want the Preview option to be available for regular reports, add 500 to the report type (in example above 511 or 2511). ***XML*** means that the report has at least one *!xml statement, and AdmReport will be run with the ***-xml*** switch. ***CSV*** means that the report has *!CSV! statements, and AdmReport will be run with the -csv switch. ***Excel*** means that the AdmReport will run with the ***-excel*** switch.
XMLSTYLE/An	Optional field containing the pathname of the style-sheet to use if a report is run with the -XML switch.
XMLOUT/An	Optional field containing the path-name of the output file (.XML file) generated by REPORT if run with the -XML switch.

M.2.4 Parameters file

The TRANS_ENV file must have an entry appmenu.parameter=FILENAME to identify the file containing information about the parameters to be entered and verified. This file must contain the following fields:

TSKID/I KEY1	Same as the TSKID in the taskfile.
PSEQ/I KEY2	A sequence number to order the parameter entries for the TSKID.
APVPATH/An	Contains the pathname to the file describing the parameter.

INSTANCE/I	If the APVPATH file is parameterized using instances this field must contain the INSTANCE number, else 0.

M.2.5 Parameter defaults file

The TRANS_ENV file may also specify appmenu.defaults=FILENAME to enable default values for parameters with the SAVE option. Any new values entered into those parameters will also be saved in this file. The file must contain the following fields:

TSKID/I KEY1	Same as the TSKID in the taskfile.
LNM/An KEY2	The logical name for which a default value is stored.
Username/An KEY3	May be blank (a company wide default), contain a MNUGRP value (a default for all users under that MNUGRP), a DEPTCODE value (a default for all users within a department) or a user name, providing default value for a specific user.
VALUE/An	The default value for this logical name.

If a record exists in this file for the selected TSKID and logical name its value will be presented in the APV Dialog Box. If a record with the user name as key exists its value will be used, if not the MNUGRP value will be used if it exists, or else the company default will be used. If no record exists for the TSKID/LNM combination no default value will be presented.

M.3 RMO interaction

If the RMO running with the screen has a field ADM$APPMENU/I defined, the RMO will get a call with M$M = '%M' when an attempt to select an APPMENU item is made, with ADM$APPMENU set to the TSKID of the task selected. The RMO can block the selection by setting ADM$APPMENU = 0 at this RMO call.

A task may also be run directly from the RMO in TRANS via the MENUBAR RMO subroutine:

```
STAT = MENUBAR(4,0,TASK)
```

See Appendix H.13.18 “MENUBAR - AppMenu Manipulations”for details.

M.4 Positioning Help Menu Entries in the Menu Bar

A Help menu item (a menu item with TSKTYP HLP or WHLP) may be placed anywhere in the menu hierarchy, e.g. as the last item under each major menu entry, but in most cases you would want to put them under the Help menu entry. To allow for placing menu items under predefined menu bar entries ADMINS has reserved the use of MNULVL1 values from 30,000 and up in the menuspec file.

The only reserved number right now is 30,900, which will add its members to the Help menu (if the Help menu already exists it will add to it, if it does not exist it will create it, and other menu items that belong to the Help menu will also be added to this entry.

In the Menu Specification file add a record with the keys

```
USERNAME  30,900  0  0
```

which should reference a TSKID record with a TSKTYP of 'MENU' and a TSKDESCR of '&Help' (or whatever text you want for the Help menu entry. All following entries with keys:

```
USERNAME  30,900  n  m
```

then appear under the Help menu.

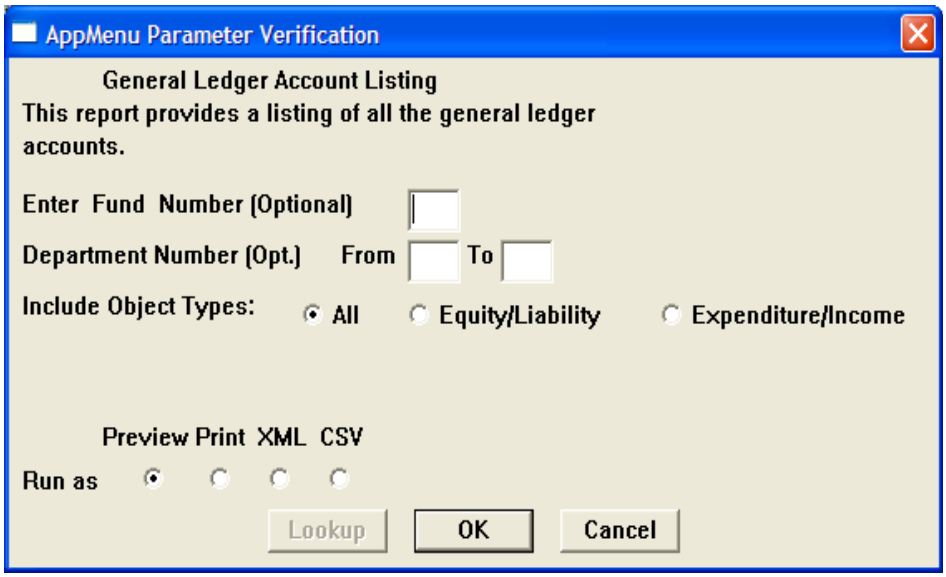
M.5 AppMenu Parameter Verification

The *AppMenu Parameter Verification* subsystem provides a mechanism to enter, verify and pass parameters to command files and reports being run from the AppMenu system. Parameters are passed to the program being run as "logical parameters", which should have names starting with "L_" and should appear between angle brackets (<>) in the program being run, e.g.

```
SELECT DEPT EQ <L_DEPARTMENT>
```

M.5.1 The AppMenu Parameter Verification Dialog Box

If the taskfile has the PVFLAG set (in the AppMenu Maintenance Utility *Create Tasks* screen its the "APV Control" checkbox), a dialog box will be opened when the user selects the task from the menu, to allow the user to enter values for the prompts, and if the task is a report, to choose how to run it.



The TASKID and the Task Title (the TSKDESC field) will display as the caption of the dialog box, instead of "Appmenu Parameter Verification" if

```
appmenu.apvcaption=1
```

is present in the TRANS_ENV file.

For reports, the "Run As" radio buttons allow you to select from the available ouput methods[1],

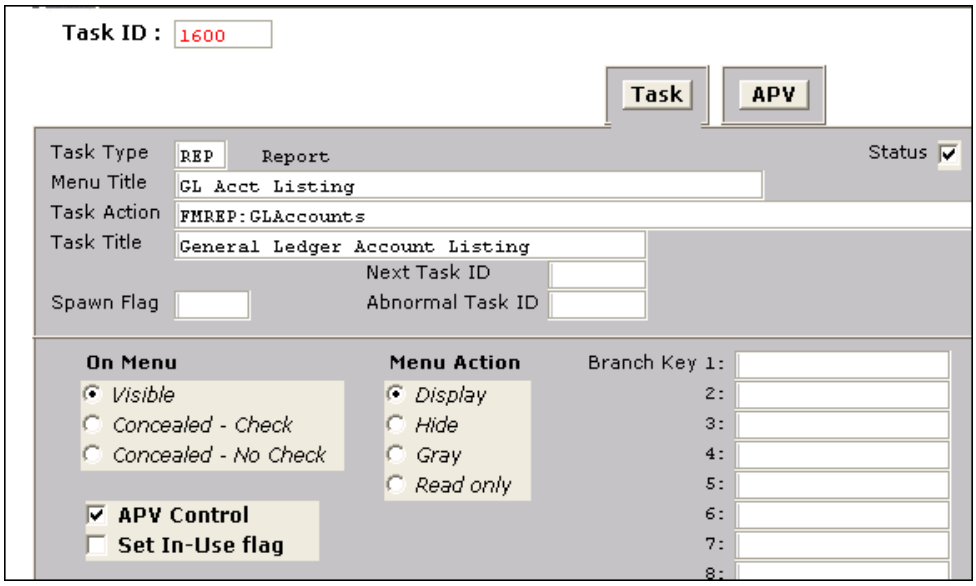
Within the dialog box, if the focus is in a field with a LOOKUP, the Lookup button will be enabled, otherwise it will be disabled (grayed). Date fields always have a month/ day LOOKUP enabled.

When the user is finished supplying the requested parameters they can press return or click OK to launch the AppMenu task . The responses will be assigned to ("L_") logical names as specified in the APV files for the task (described below in Appendix M.5.2 "APV Syntax"). In addition, the AppMenu TASKID for the task will be assigned to the logical name L_APV_TSKID.

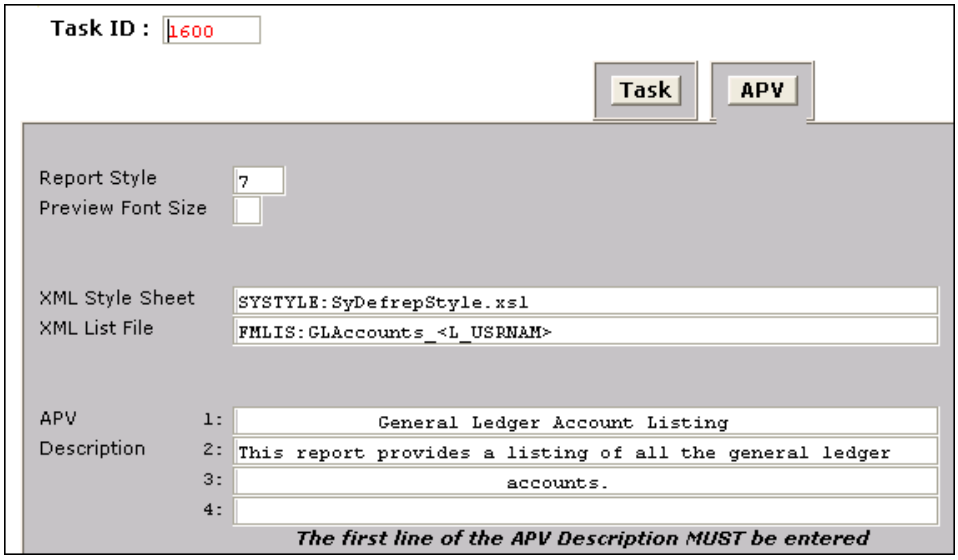
Here's how you use AppMenu to set up the dialog box shown above:

1.See REPTYP in Appendix M.2.3.1 "Additional Task file fields (for APV support)" and Appendix M.5.5 "Reports That Need a Command File".

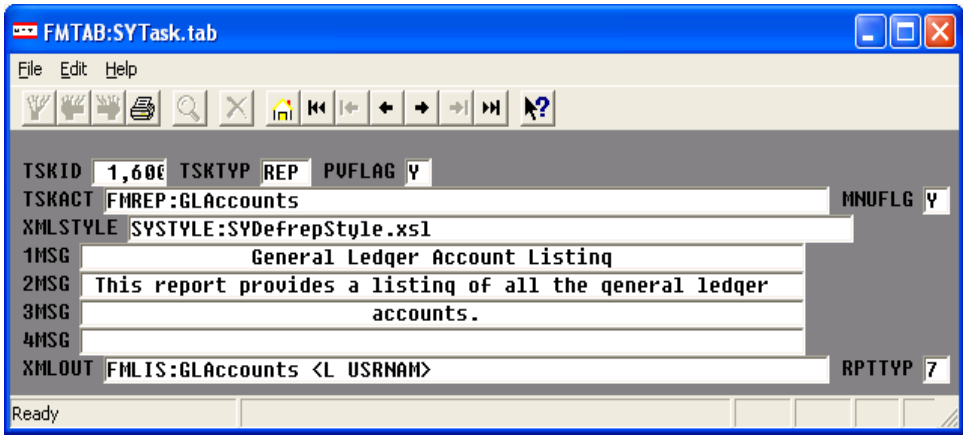
The Create Tasks screen has an entry like this:



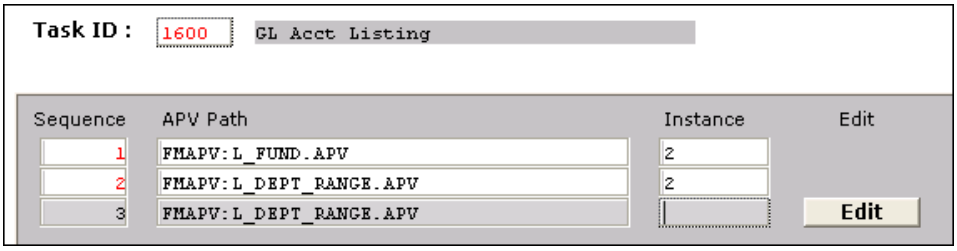
The APV pane (click the APV button) contains the following:



If you do not choose to use the AppMenu Maintenance Utility, the taskfile record resulting from the above entries would look like this:



The contents of the parameters file can be entered and viewed in the AppMenu Maintenance Utility by clicking  in the ***Create Tasks*** screen:



In this screen click  to view or change the APV files.The three .apv files used to provide specifications for the input parameters of our sample APV Dialog box might look like this:

L_FUND.apv:

```
L_FUND %1%
  format @FUND
  %2%
  %3%
Instances
!--------
! Instance 1: Required Fund Number
! -----------
1-%1%: Enter Fund Number
1-%2%:
1-%3%:
! Instance 2: Optional Fund Number
! -----------
2-%1%: Enter  Fund Number  (Optional)
2-%2%: OPTIONAL
2-%3%:
! Instance 3: Repetitive Optional Fund Number
! -----------
3-%1%: Enter  Fund Number  (Optional)
3-%2%: OPTIONAL
3-%3%: REPEAT
```

L_DEPT_RANGE.apv:

```
L_DEPT_RANGE %1%
  format @DEPT
  range LO From
  range HI To
  %2%
Instances
!--------
! Instance 1: Required Department Numbers
! -----------
1-%1%: Department Number
1-%2%:
! Instance 2: Optional Department Numbers
! -----------
2-%1%: Department Number (Opt.)
2-%2%: OPTIONAL
```

L_OBJTYPE.apv:

```
L_OBJTYPE Include Object Types:
  format @OBJECT
  radiobutton All/'gt 0' 'Equity/Liability'/'bet 1000 and 2999'
  radiobutton 'Expenditure/Income'/'ge 3000'
```

M.5.2 APV Syntax

The APVPATH field in the *appmenu.parameter* file contains the pathname of a text file specifying the parameter to be verified, e.g. SYAPV:L_DEPT.apv. For documentation purposes it is our recommendation that this file is named the same as the logical name or parameter name it will create, with the .apv file type (but it may be named any way you want).

The syntax of the APV file is:

```
LogicalName Prompt text
KEYWORD Value
KEYWORD Value

…
```

Where "LogicalName" is the name of the parameter to be verified[2]. Generally its good programming practice to use a name that starts with "L_" or "L$" because:

1. The logical name must start with L_ (or L$) to support automatic substitution in "logical parameters" (<L_name>) in reports and command files.
2. Certain APV file statements (described below) will refer to the value returned by another APV file if a parameter name is supplied that begins with "L_" or " L$".

"Prompt text" can either be a literal value (e.g. "Start Date") or may include the token "***%label%***" (e.g. "Enter %label%") which tells AppMenu to substitute the label from the associated element in the data dictionary. When using this token the FORMAT for the APV must use a data dictionary reference, for example:

```
FORMAT @ELEMENT_NAME
```

or the prompt text will include the literal "%label%").

2.The parameter will be assigned to a logical name (when used in the AppMenu subsystem) or to a logical name and/or a field in the RMO (when used with the APVDLG subroutine (see Appendix H.13.17 "APVDLG: Activate an APV Dialog Box From the RMO")

Currently the following keywords are defined:

FORMAT *data type*	Can be a valid ADMINS data type (e.g. X9999) or reference a field name in the Data Dictionary, e.g. @DEPCODE. If present the entered value will be checked that it's valid for the data type. If the FORMAT is given as a reference to a Data Dictionary field, and this field is tied to a code list which has a lookup defined in the dictionary, this lookup will be available to the user. Also, if the referenced DD element has CAPS or LOWERCASE specified the values entered will be rendered accordingly.
XPOS *column*	Start edit box etc. in position *column*.
CAPS	Capitalize the entered value.
LOWERCASE	Lowercase the entered value.
NOECHO	Do not echo the entered characters (e.g. password)
REPEAT	Repetitive parameter. Logical names L_name, L_name2, etc. is created until the user stops entering values.
OPTIONAL	The user does not have to enter a value. If no value is entered, a logical name with the value 'CR' (or '%CR%' if OPTION 5 is in effect) is created.
SAVE [DEPARTMENT| GROUP|USER] [/NOREUSE]	The value of the logical name is saved in the defaults file, with username blank if SAVE only, username set to the MNUGRP value if SAVE GROUP, or username set to the users login name if SAVE USER. The /NOREUSE switch may be used to save the users entry in the defaults file, but not to use it as a start value next time.

RANGE lmod Prompt	The RANGE keyword must always appear in pairs to specify prompt strings and logical name modifier for the low and high values. If you e.g. have:
	`L_DEPT Enter department range`
	`  RANGE LOW '1st department'`
	`  RANGE HIH 'Last department'`
	the logical names L_DEPTLOW and L_DEPTHIH would be created.
	You may also specify the logical name ending in _RANGE, e.g.
	`L_DEPT_RANGE Enter department range`
	`  RANGE 1 '1st department'`
	`  RANGE 2 'Last department'`
	in which case the logical names L_DEPT_1 and L_DEPT_2 will be created.
	
	To include the label attribute of the data dictionary element specified in the format statement into the prompt string use the "***%label%***" token, e.g.
	`L_DEPT_RANGE Enter %label% range`
	`  RANGE 1 1st %label%`
	`  RANGE 2 2nd %label%`

RADIOBUTTON Label[/Value] Label [/value] …		Instead of prompting the user with an edit box the user will be presented a number of radio buttons to choose one value. If a label is followed by a /value the value is what is assigned to the logical name if this button is clicked, else the Label itself is used as the value.
		Labels may be up to 31 characcters long. If the Label (or the value) contains spaces it must be enclosed in quotes, e.g.
		`RADIOBUTTON 'My Label'/1 'Your Label'/2`
		
		To designate a default value in case no value has been saved from a previous run insert a '!' between the '/' and the actual value, e.g.
		`RADIOBUTTON Small/'< 100' Medium/!'Bet 100 and 200' Big/'> 200'`
		In this case Medium is by default checked if no saved value is available.
		RADIOBUTTON specifications that cause "hard-to-read" long lines may be continued on the next line (but a label/ value pair must fit on a single line)
		To include the label attribute of the data dictionary element specified in the format statement into the prompt string use the "***%label%***" token.

NOCODELISTCHECK		Do not check the entered value against the format field's code-list.
ERRORIFINCODELIST		Check entered value against the code-list, and make it an error if it is present in the code-list.
		Note: *date fields have a "calendar" lookup by default*
LOOKUP *filename*		Path of file for LOOKUP[a]
	LKUP TITLE	Title string (can include %label% token)
	LKUP HEADING	Heading string (can include %label% token)
	LKUP FOOTING	Footing string
	LKUP DETAILBREAK	KeyFieldName
	LKUP LINK	=prefix link-file KEY IS | KEYS ARE key-field … (keys may be G$fields)
	LKUP CREATE	Fieldname/type expression
	LKUP SELECT	Selection clause The SELECT clause may reference fields in the LOOKUP virtual record, constants, and G$ fields in the screen's virtual record. If G$ fields are used it is the developer's responsibility to make sure that those G$ fields are present in all screens where the APV dialog may be activated.
	LKUP KEY_RANGE	low-key high-key (may be G$fields) typing anything will launch LOOKUP
	LKUP BOUND	entry interpeted as entry value for LOOKUP
	LKUP BOUND_KEY	List of field names (separated by space). May be G$fields)
	LKUP DISPLAY	Uppercase|Capitalize first character (in RETURN or TRANSFER value)
	LKUP CAPS|CAP1	
	LKUP NOLOCKEDRECORDS	Fieldname INTO Logical-name
	LKUP RETURN	
	LKUP TRANSFER	
NODISPLAY		Do not show this parameter entry in the APV dialog box.

USE *fieldname* FROM *filename* KEY IS | KEYS ARE *key-value …*	Open the file *filename* and use the provided key values to find a record, and use the value in that record's *fieldname* as the parameter value. *Key-value …* are one or more constants or G$fieldnames that must satisfy the data type of the corresponding key field in filename.
USE VALUE *constant*	to set *constant* as the default value.
DEFAULT *fieldname* [*fieldname_high*]	Field names that contain default values to use.
EXIST IN *filename* KEY IS|KEYS ARE *key-value …*	A record with the provided key values (may be G$ field) must exist in the file.
NOEXIST IN *filename* KEY IS|KEYS ARE *key-value …*	It is an error if a record with the provided key values(may be G$ field) already exists in the file.
DISPLAY *fieldname* FROM *filename* KEY IS | KEYS ARE *key-value …*	Open the file *filename* and use the provided key values to find a record, and display the value of that record's *fieldname* next to the parameter value. Key-value … are one or more defined parameter or constants that must satisfy the data type of the corresponding key field in filename. fieldname/nn may be used to specify the display width to use for the field. If not specified the default display width for the fields data type will be used.
DISPLAY D% DISPLAY D%/nn	Display the Description field from the internal codelist tied to the field which the FORMAT @FIELDNAME statement establishes. The standard width of the D% field is 60 characters, which may be modified using the "/nn" syntax, where the nn is the number of characters to display (because a variable font is used, the number of characters actually displayed may vary, the 'nn' represents an average width of 'nn' characters). The display statement may only be used for single edit boxes, not for ranges or radiobuttons.
OPENFILEDIALOG Start-directory OFD FILTER Filter-spec. OFD TITLE Dialog box title	Start-directory can be an absolute directory path, a logical name containing a directory path, or the name of a previously declared APV parameter which's value contains a directory path (absolute or as a logical name). Filter-spec. is any pairs of text and wildcards separated by a '~' (tilde), as described in the ADMINS Procedures Manual section H.13.1.1 File Open Dialog Box, e.g. 'My csv files~*.csv~My text files~*.txt;*.lis' Dialog box title is any text you want to appear in the title bar (caption) of the dialog box. If not present the default Open File Dialog Box title will appear.

CHECK logic statement
Error Message

The CHECK statement basically follows the same rule as 'C' statements in TRANS. You reference the current and previously defined parameters by their logical name, e.g.

```
CHECK L_R_DATE2 GE L_R_DATE1
Date 2 must be after or the same as Date 1
```

The Logic Satement may be continued on the next line by placing a colon (:) at the end of the line.

To allow data to be entered in a "random" order all CHECK statements are executed after the user presses the [OK] button.

You may also reference global fields (G$ fields) and TODAY in a CHECK statement, but remember, if you do, these fields have to be present in every screen the APV dialog can be run from.

The CHECK statement may include subroutines, e.g.

```
CHECK CHKDATE(L_R_DATE,TODAY,30,90) NE 1
```

which will give an error message if the entered L_R_DATE is more than 30 days prior to TODAY or more than 90 days later than TODAY.

Observe that only subroutines that return an integer status value that can be tested can be used in this way.

a.If FORMAT is given as @FIELDNAME (that is, use the format of a field in the Data Dictionary) and this field is tied to a code list with a lookup defined, this lookup will automatically be available. If no such lookup is available for the parameter you may define a lookup using the APV LOOKUP syntax.

M.5.3 Instances: Handling different uses of a logical parameter

To make an .APV file for a given logical parameter support e.g. both required and optional parameters (e.g. <L_DEPT> and <<L_DEPT>>) we support specifying all or parts of an APV line as a token to be substituted at run-time, depending on which "instance" is specified. Place the token in the code as **%n%**, where n is any number. How it works is best explained through an example:

Consider the following L_DEPT.APV file:

```
L_DEPT %1%
  FORMAT @DEPT
  %2%
INSTANCES
1-%1%: Enter Department Number
1-%2%:
2-%1%: Enter Department Number (optional)
2-%2%: OPTIONAL
```

Under INSTANCES the first number identifies the instance number, and the second number identifies the token number under that instance. The rest of the line is what will be substituted for the token in the APV file syntax when that instance is specified.

In the *appmenu.parameter* file an instance supporting a required <L_DEPT> parameter would reference the APV file as:

```
L_DEPT.APV 1
```

and AppMenu would interpret the APV for this instance as if it read as follows:

```
L_DEPT Enter Department Number
  FORMAT @DEPT
```

(The values for Instance 1 are substitued for the two tokens)

an instance supporting an optional <<L_DEPT>> parameter would reference the APV file as:

```
L_DEPT.APV 2
```

and AppMenu would interpret the APV for this instance as if it read as follows:

```
L_DEPT Enter Department Number (optional)
  FORMAT @DEPT
  OPTIONAL
```

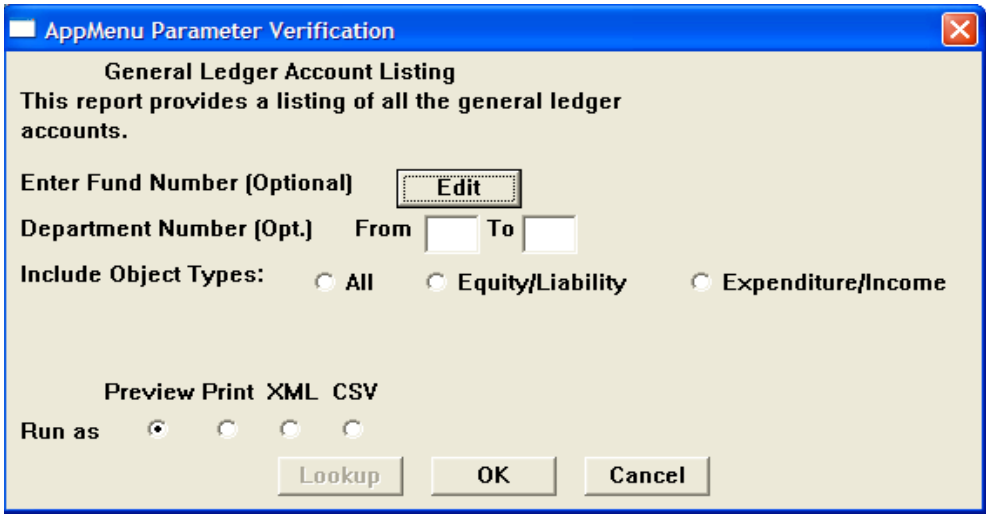
(The values for Instance 2 are substitued for the two tokens)

M.5.4 Repetitive Parameters

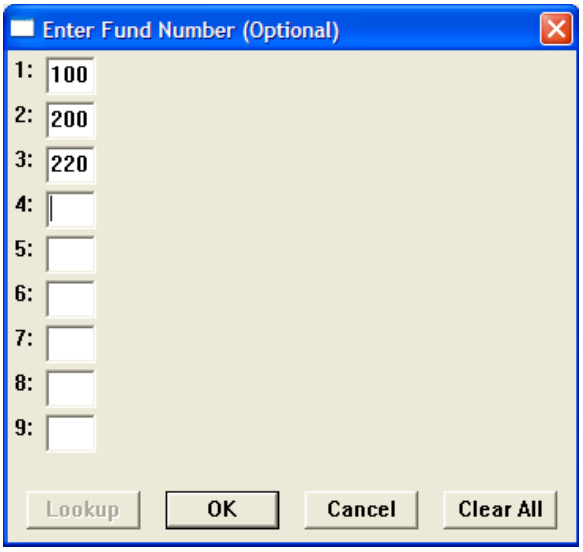
Repetitive parameters, specified in the report or command file using the '~' (tilde) syntax, e.g.

```
<<L_FUND>>~
```

repeatedly translate a series of logical names L_FUND1, L_FUND2, … until L_FUNDn is not found, or it contains a "CR" (or "%CR%" if option "5" is in effect), or until the ninth logical name in the series is found (i.e. L_FUND9 is the last logical name in the L_FUNDn series that is checked). Since up to 9 values may be entered a separate dialog box is used to view and enter the values, and in the main dialog box a push-button labeled Edit will appear:



When the Edit button is pressed a dialog box with nine edit boxes will appear.



If the parameter has the SAVE option the values from the previous run will be presented, and any changed value will be saved.

M.5.5 Reports That Need a Command File

Often parameterized reports are run as a step in a command file, with the command file being the task that the user selects from AppMenu. One common situation is perhaps a number of ADMINS commands like MOVE, PROD, MAINT etc. are used to prepare the data that is to be reported. Like reports called directly from AppMenu, reports called in a command file can use APV to load and check logical parameters and to select from the available output types for a "multi-purpose report".

To accommodate this situation put the report specifications (e.g. *XMLSTYLE*, *XMLOUT*, *RPTTYP* etc.) and logical parameter APVs in the taskfile (the *Create Task*s screen) entry for the command file (*TSKTYP* = '*COM*') just as you would if the report was being called directly from AppMenu.

If you e.g. intend to run, for example, *MY_REP:MyReport.rep* in this manner your command file should contain the code:

```
<L_APV_REPCMD> MY_REP:MyReport
```

The APV Subsystem will create the logical name L_APV_REPCMD with the appropriate parameters to run the report in the desired way. E.g. if you select to run the report as XML the value of the L_APV_REPCMD logical name might be set by AppMenu to:

```
AdmReport "-xml=MY_LIS:MyReport" -browse
```

so that the command file would end up running with this line to call the report (after parameter substitution):

```
AdmReport "-xml=MY_LIS:MyReport" -browse MY_REP:MyReport
```

M.6 Miscellaneous options

The following optional behaviors have been implemented.

M.6.1 Handling the current screen as a menu item

By default, if a menu item is the current screen, it will not show up when the menu is opened. The TRANS_ENV file may be used to modify this behavior:

```
appmenu.currentscreen=gray | on | off
```

Off is the default. *On* makes it show up like a regular menu item, while gray makes it show up in gray state (disabled).

M.6.2 Viewing Report Output

When running reports from AppMenu one of the output options normally is Preview. This causes the report to be run with the "***/html/browse***" switches, causing the report to be produced with the HTML page wrappers, and discarding any style escape sequences intended for the printer. This has the side effect that very wide reports will not print correctly from the browser (long lines are being truncated or wrapped).

Placing

```
Appmenu.reportviewer=TedRE
```

in the TRANS_ENV file changes this default behavior. AppMenu will run the report with the /view switch and invoke TedRE to view the resulting .LIS file. When printing from this viewer the printed version will look exactly as if the report had been printed directly from AdmReport (you can print wider pages and style escape sequences are retained).

Appendix N:

(This chapter's content only appears in the OpenVMS version of the Manual.)

Appendix O: Obsolete Commands and Syntax

1	(option) Terminal is VT100 terminal. ADMINS now checks terminal type via a system service.
2	(option) Terminal is VT52 terminal (see above).
4	(option) Enabled TRANS to use color REGIS graphics on VT241 terminals.
A	(option) Specified "automatic decimal point alignment" in arithmetic computations. This is now the default behaviour. "Manual" (i.e. application-controlled) decimal point alignment can be specified by including "." in the string assigned to the logical name OPTION (see Appendix A: "Options").
[ADDER]	(directory) Directory that held multi-adder control files.
ADD	(keyword) File in TRANS was to have "multi-adder" protection.
ADM$KBBOARD	(logical name) Formerly used to designate the "soft keyboard" table file. This functionality has been superseded by the TRANS environment file (see Section 6.17 "The TRANS Environment File")
ADM$MENU_KEY	(logical name) Formerly used to redefine the keystroke that activates the menu bar. Superseded by the TRANS environment file.
ADM$NO_FCC	(logical name) Formerly used to cause ADMINS commands that produce ADMINSxx.LIS output files to produce Stream_LF files instead of FORTRAN carriage control files.
ADM$PAUSE	(logical name) Formerly used to cause TRANS to pause the specified number of hundreths of a second before trying to read input when it has just output more than one character.
ADM$QUERY_KEY	(logical name) Formerly used to redefine the keystroke that activates the QUERY by TRANS. QUERY by TRANS is no longer supported.
ADM$RNF	(logical name) Formerly used to identify file containing lookup table of control sequences, etc. for use with RNF ".cc" or ".lk" syntax. Superseded by logical name ADM$STYLE, which is used by both RNF and REPORT.
ADM$SCR_KEY	(logical name) Formerly used to redefine the keystroke that activates the subscreen menu. Superseded by the TRANS environment file.
B	(option) or:

-B	(appended to file-spec) Bypassed file protection.
CALC	(ANALYZER statement) Superseded by CREATE.
CLR.EXE	(command) ADMINS command that cleared the video screen and placed the cursor at the top left corner. Superseded by the DCL command file ADM$DIST:CLR.COM. (The symbol CLR in ADMSYMDEF.COM now points to @ADM$DIST:CLR.)
d (lowercase)	(option) Early (pre-Version 3.0) versions of TRANS would dump the DA array at a EXIT TO MINIMIZE FILE DAMAGE event.
DIFFDA	(subroutine) Returns the difference in days between two dates. Superseded by TMDIFF (see Appendix H.4.1 "TMDIFF - Difference Between Dates and Times"). Syntax: `STAT =DIFFDA(DATE1,DATE2[,RESULT])` If RESULT is not present in the list of arguments the result of DATE2 - DATE1 is loaded into STAT.
DIFFTM	(subroutine) Returns the difference in seconds between two times of day. Superseded by TMDIFF (see Appendix H.4.1 "TMDIFF - Difference Between Dates and Times"). Syntax: `STAT =DIFFTM(TIME1,TIME2[,RESULT])`
FINDREC	(subroutine) Find and read record specified in file specified. Superseded by RECOPN and RECIDX subroutines (see Appendix H.10.2 "RECOPN and RECIDX - Access Records in any File").
group event flags	ADMINS functionality that used group event flags is now implemented using the VMS distributed lock manager.
HR	(subroutine) Used to assign a series of logical names at one time.
I	(option) Block Overlap Protection (now implicit)
KEYSTROKE	(keyword) Former Menu Bar paragraph optional keyword to redefine the keystroke that activates the menu bar. Superseded by the TRANS environment file.
M	(option) Enabled "monitoring": information for the major ADMINS processing command would be placed in the process name. Now ignored.
MOVLNK	(subroutine) Automatically moves a series of fields in TRANS from one main file or link file record to another main file or link file record. The MOVFLD subroutine, described in Appendix H.13.15 "MOVFLD - Move Fields Among Files Accessed via TRO", has superseded MOVLNK. Syntax: `STAT = MOVLNK(FROMFILE,TOFILE,FRMFLD,TOFLD,#FLDS)`
MULTI	(command) Managed multi-adder file, cleared files-left open, flags left on, etc.

O	(option) Invoked "Character Display Optimization" in TRANS. Now ignored.
PRT and PKB	(command) ADMINS print ASCII text file commands. Superseded by operating system printing functionality.
q (lowercase)	(option) Formerly used to modify how TRANS displayed screens. TRANS screen i/o has been reimplemented and this option is ignored.
Query by TRANS	Mode of TRANS in which user could enter selection criteria, then TRANS would only display records that meet selection criteria. Because this facility could perform only sequential searches its performance was unacceptably slow, and consquently it has been dropped.
REGIS	Graphics interpreter available on certain DEC terminals. Supported by the now obsolete ADMINS Color Business Graphics subroutines: BAR, HBAR, BRAT, HBRAT, LINE, LRAT, PCHART, REGIS, GICLR, GITXT.
RLKOUT	(keyword) Requested record-lockout protection for file (now implicit).
RNF	(command)Document Runoff facility
SEQ	(command) Superseded by FILECONVERT (see Section 13.4 "FILECONVERT - Convert ADMINS datafile attributes"), which has identical syntax for the sequentialize operation.
SCRED	(command) Superseded by TED (see Appendix J: "The TED Text Editor").
SPROD	(command) Special version of PROD optimized for circumstance when detail file was in sort on link fields. Functionality now implicit in PROD if detail file is in sort on link fields.
SH$field	Group shared area fields allowed ADMINS applications to utilize the Group-shared area feature of single-node OpenVMS systems. Also "emulated" by ADMINS on OpenVMS clusters and Windows for compatibility.
SYS$DEVICE	(logical name) Identified disk on which multi-adder directory ([ADDER]) is located.
U	(option) Restricted (to SYSTEM only) ability to use MULTI command to clear files.
XC	(subroutine) Formerly used to reverse the 16-bit words in D and F fields, in situations when ADMINS received the data with the words "switched". This situation no longer rises, so XC is no longer needed. For compatibility, if XC is called it simply moves the data to a new location without changing it.
z (lowercase)	(option) Changed RNF so that it output the same number of lines in the same area of the page on page 1 as it did on all subsequent pages. Now RNF always works this way.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

M

N

O

P

Q

R

S

T

u

v

w

X

Y

Z