TABLE OF CONTENTS

Chapter 1
INTRODUCTION TO UTILITIES

ABOUT UTILITIES

This manual describes the various utility programs that come with CfMC software. Utilities perform common tasks, such as copying or converting data files or creating reports about your data. These utilities allow you to execute complex tasks without learning the underlying commands and syntax of Survent or Mentor.

They often have a standard user interface that includes menus and fields for you to fill in. Many provide a final screen that summarizes your choices and gives you the option to exit before proceeding. You can also save a command file to run again in "batch" mode later.

The utilities described in this manual are programs that can be used by either Survent or Mentor users. Utilities specific to Survent (such as Quotamod, Suspres, Foneutil and Fonebuld) are documented the Survent manual.

The utilities described in this manual fall under four categories:

File management	Data Analysis	Data Alteration	Customization
COPYFILE	HOLE	CLEANIT	CFMCMENU
DBUTIL	LIST	MERGE	MAKEMSG
MAKECASE	SCAN	RECODE	
MAKEVARS			
MERGE			
RAWCOPY			
REFORMAT			

The Appendices

This manual's appendices include other items of interest to both Survent and Mentor users:

- A comprehensive listing of "meta" commands (commands used across multiple programs)
- A glossary of CfMC terms
- An overview of CfMC programming conventions
- Information about file transfers
- A chart of ASCII graphic characters and their hexadecimal values
- A list of CON and ZSPC statements (for accessing system information)

UTILITY CONVENTIONS

Many utilities share several common features in terms of screen navigation and file names.

Screen Navigation

Menus. On menus, enter the letter of your selection or use the arrow keys (or Ctrl-U and Ctrl-D on UNIX terminals) to highlight your selection and press Enter.

To back up to the previous screen enter a caret (^) and press Enter.

Fields. Enter responses at the arrow prompt (-->). If there are dots, they represent the number of characters that can be entered in that field. You can use the BACKSPACE key to delete text. Press Enter when you are satisfied with your response. Multiple responses can be entered in this field separated by commas (for example: 01,07,03). To back up to the previous screen from an arrow prompt, enter a caret (^) and press Enter.

When the field is a box, enter text and then press ESC. You may use the arrow keys to move the cursor, and use the BACKSPACE and INSERT keys to correct text. If the box is more than one line, there is no need to press Enter at the end of a line; text will automatically wrap. (UNIX users: You can only correct text on the current line you are on. You must press Enter until you reach a > prompt to move on to the next screen.)

To back up to the previous screen from a box field, enter a caret (^) as the first and only character and press ESC.

File Names. When you are asked to specify a file name, you can enter the name (i.e. BANK) to indicate the file is in the current directory, or if the CfMC variable

is set, the utility will look in that directory. You can also enter a full or relative path name (i.e. c:\J123\BANK or ../j124/bank) to indicate the file is in another directory. When you enter a study name, do not specify the standard datafile TR extension; it is assumed.

> **NOTE** *Be sure to use unique output file names. If you run the same utility twice with the same output file name, it will rename the first output file.*

PLATFORM DIFFERENCES

Survent and Mentor operate in the DOS (IBM-PC compatible), and unix (linux, aix, hpux, etc.) environments. Unix filenames must be lower case to be read by CfMC programs. DOS filenames cannot be longer than eight characters with a threecharacter extension CfMC file types have unique extensions.

These differences are summarized in the following chart.

Operating system	Computer Hardware	File structure	File names
DOS	IBM PC or compatible	\ACME\A123\	BANK.TR
UNIX	UNIX PC or minicomputer	/acme/a123	bank.tr

Only ASCII files and CfMC .tr data files are compatible across all platforms. All other files created by CfMC software can only be used on the platform that they are generated on (.fon, .db, .qff, .quo, suspend files, etc).

MAXIMUM NUMBER OF TABLES

CfMC utilities use the local default database setting when creating tables. To adjust the maximum number of tables the utilities can create, use the meta command >DB_SIZES in your mentinit file. To allow approximately 5,000 tables per run, use the setting:

```
>DB_SIZES= 0 0, 0 0, 0 0, 50000 0, 0 0
```

The >DB_SIZES command is described in detail in *Appendix A: Meta Commands* of this manual.

WHERE DO I GO FROM HERE?

We hope you find this manual useful. CfMC is always looking for feedback about our software, our manuals and our technical support.

Below is a list of ways to contact us, depending upon your needs.

IF YOU WANT TO:	CONTACT	VOICEMAIL	E-MAIL/WEB SITE
Get general info.	Receptionist	(415) 777-0470	info@cfmc.com www.cfmc.com
Lease CfMC software, add more users, upgrade	Marketing	East: (415) 536-2774	joycer@cfmc.com
		West: (415) 777-0470	sales@cfmc.com
Get help with using the software, have a suggestion, want a new feature added to software, want to report a bug	Tech Support	(415) 777-0470	support@cfmc.com (e-mail preferred)
Get training	Training	(415) 777-0470	train@cfmc.com
Get copies of manuals, have corrections for the documentation	Documentation	(415) 777-0470	doc@cfmc.com
	CfMC-S.F. Web site		www.cfmc.com
Talk to other users of CfMC software	Spec-talk discussion group		spec-talk@cfmc.com
Get a quote from the San Francisco Service Bureau	San Francisco office	(415) 777-0470	servbur@cfmc.com
	or CfMC-SF Web site		www.cfmc.com

IF YOU WANT TO:	CONTACT	VOICEMAIL	E-MAIL/WEB SITE
Get a quote from the Denver Service Bureau	Denver office	(303) 860-1811	denver@cfmc.com
	or CfMC-Denver Web site		www.cfmc.com/denver

Chapter 2
FILE MANAGEMENT

2.1 COPYFILE

What it does

COPYFILE allows you to do basic file management, such as copying, sorting and printing data files by selecting options from menus. Since COPYFILE is menu driven, it allows you to do sophisticated operations to data files without having to know Mentor command syntax. If you want to learn Mentor command syntax, the underlying commands for COPYFILE are included at the end of this chapter.

When to use it

Use COPYFILE to copy, subset, translate, sort or combine data files. If you need to move data between files, use MERGE. If you are trying to build a CfMC dataset and have a variable numbers of records per case, or missing records, use MAKECASE instead of COPYFILE.

How to use it

At the operating system prompt, enter:

```
COPYFILE
```

You will have to supply different information depending on the task you select (see *OPTIONS* below). You can press ESC to exit most screens, or enter `terminate` in a text field to quit COPYFILE. When you have completed all the screens, COPYFILE displays a summary screen which indicates the options you have selected and it gives you a chance to change them before you run the operation.

COPYFILE will return to the main menu screen when it has completed the task you have selected.

Options

1 COPY

Imports or exports several different file types. You can also copy a file, change record length, or attempt to fix a corrupted file. You can use the following file types as input:

- CfMC System (TR)
- ASCII
- ASCII Card-image (multiple records per case)
- ASCII Delimited file
- Binary
- Swapped Binary (from other types of computers)
- CfMC HP3000 SPL format (DTA)
- CfMC Phone file (FON file from FONEBULD)

You can have the following types of files as output:

- CfMC System (TR)
- ASCII
- ASCII Card-image (multiple records per case)
- ASCII with blanks removed from end of records
- Binary
- Swapped Binary (for other types of computers)
- CfMC HP3000 SPL format (DTA)

You must indicate a record length between 2-65000 columns. ASCII, card image- and binary files must have a record length that is a multiple of 80. When writing out a standard CfMC TR file, you can also index the file or try to fix a corrupted file (usually one that was previously left open).

Copying from a delimited file produces a called DELIM.MAP which contains a map of locations and lengths written to and, if the field is numeric, the mean, standard deviation, standard error, minimum value and maximum value.

Only standard CfMC TR files can be exported to CfMC HP3000 SPL (DTA) format.

2 COMBINE

Combines two or more CfMC System files. You can create a new file or append to an existing file. The output file case length must be greater than or equal to the longest file being combined and less than 65,000 columns. You can combine up to

a total of 999,999 cases.

You can enter each file name of each file you want to use, or you can use “wild cards” to select a group of files. The wild card characters are “*” to match any character, “?” to match single alphabetic characters, and “#” to match single numeric characters. For example, “RRUN##” would match any file names that start with RRUN and end with any two numeric characters, such as RRUN05, RRUN10, and RRUN27.

Use the MERGE utility to combine cases from two files and/or make longer records.

3 SORT

Sorts a CfMC System file. COPYFILE will sort the file sequentially based on the data in the variable or location [column and row(s)] you provide. This is called a *sort key*. You can provide up to five sort keys for a file.

You can also do a random sort (this is useful for scrambling phone numbers in a FON file), by using the Random_Value keyword for the sort key.

4 SUBSET

Allows you to create a file that contains a subset of an existing CfMC system file. This option writes multiple files with every "Nth" record

You can also write one or many files with every "Nth" record in the file or files. This is often used to create a subset of sample files in order to distribute them to other interviewing houses. For instance, you can either write 10 files with the 1st, 2nd, etc. record going to each file or just write one output file with the 1st, 11th, 21st, etc. record in it.

Another example: if there are 1,000 records and 100 random records are supposed to be sent to each of ten interviewing houses, the utility will put the 1st, 11th, 21st, 31st, etc., records into file No. 1; the 2nd, 12th, 22nd, 32nd., etc., into file No. 2, and so on. Thus, a random sampling is distributed.

You can select:

- cases based on data criteria
- a random number or percentage of cases
- the first 'n' number of cases

SUBSET will offer to write a second data file that contains all the data cases not written to the first data file. To select cases based on data criteria, you can enter either a data location or use variables defined in a DB file.

5 PRINT

Prints the data from a CfMC system file into an ASCII file. The output file contains the case ID and the data from each case. You can choose the number of cases to print and can specify how many 80-column records from each case to include. The output file has the default extension of PRT.

6 DISPLAY

Displays a listing of CfMC data files. This includes all files with the following extensions:

ASC BIN SWB TR

These represent ascii, binary, swapped binary, and CfMC standard data files.

7 TEXT

Allows you to display text data from CfMC System files. You can also move text pointers or move text from one location in the data file to another. You must know the column location of where the text area starts in the file.

Advanced Users Note: If you want to learn Mentor command syntax so you can manage files without using COPYFILE, see *Mentor: TILDE COMMANDS* and refer to the following commands:

~ADJUST
~CLEANER SAY and DISPLAY_ASCII
~DEFINE PROC= WRITE_CASE and PRINT and MODIFY
~EXECUTE PROC=
~INPUT
~OUTPUT
~REFORMAT
~SORT

2.2 MAKECASE

What it does

MAKECASE converts an ASCII or binary file into a CfMC System file (TR).

When to use it

Mentor and some CfMC Utilities require data to be in a CfMC system file, so use

MAKECASE if your data is not in a system file (for example, if it has not been collected by Survent). Since MAKECASE checks the number of cards per record and matches case IDs, use it instead of COPYFILE if you have variable number of records per case or if you suspect you may have missing records in your data file.

MAKECASE requires that the input file have data records 80 columns long. ASCII files greater than 80 columns long must be translated using the COPYFILE utility.

MAKECASE allows for a maximum of 50 records per case. If you have a file with more than 50 records per case, you will have to construct separate files and then use the MERGE utility to combine them.

How to use it

At the operating system prompt, enter:

```
MAKECASE
```

At a minimum, you must provide an input file name, file type, a case ID location (up to ten columns wide), and a record ID location (up to two columns wide).

Most data files will have a case ID in front of each record and then a record ID immediately after the case ID or at the end of the record. If a file has case IDs but no record IDs, MAKECASE will assume there is only one record per case and will construct the system file with record one as case one, record two as case two, etc. If you have a file that has alphabetic record IDs, you must change them to numeric, and MAKECASE has an option to let you do so.

You can press ESC to exit most screens, or enter `terminate` in a text field to quit MAKECASE.

When you have completed all the screens, MAKECASE displays a summary screen which indicates the options you have selected and gives you a chance to change them before you run the operation.

Input Files

MAKECASE accepts the following file types:

- ASCII
- Binary
- Swapped Binary
- CfMC System (TR)

Output Files

Below are the default file names for the files produced by MAKECASE. Remem-

ber that if you are going to run MAKECASE repeatly, you will want to assign unique file names so you do not overwrite existing files.

Name	Ext.	Description
SYSF1	TR	data file
CASE	ERR	print file
SNGL	TR	single-record System file (used to make sure the cases have been constructed correctly. You can CLEANIT on this file to look at it.)

Sample Output

The SYSF1 file will be a System file with only good cases in it. CASE.ERR is a file that contains errors MAKECASE generates when it encounters the following types of records:

```
BAD: does not have a known record ID.
```

MAKECASE will print an error message that includes the case ID, the bad record ID and the contents of the bad record.

```
DUPLICATE: a record that has the same case ID and record ID as a
```
previous record.

MAKECASE will print an error message that includes the case ID, record ID, and the contents of the duplicate records. The last duplicate record will be the one kept for the case.

```
MISSING: a case that is missing a record.
```

MAKECASE will print an error message indicating that the case will be dropped from the output file. (You can designate certain records as not required for a case; these records will not generate an error message.)

The following is a sample CASE.ERR file. The case ID is in columns 1-4 and

the record ID is in columns 79-80.

```
First case is ID 0001
Rec # 25 has a bad record ID ' ' -- case ID = 0007
0007 45871562 37459277 9827 7626
case ID 0007 is missing record 03
case ID 0007 dropped because of missing record(s)
case ID 0009 -- Record 03 is duped -- Rec # 35 -- Last record kept
00093 5002 355 33867691 166 1214 03
00093 35566832 80549879 7388 5524 03
case ID 0009 is missing record 02
case ID 0009 dropped because of missing record(s)
Last case is ID 0010
Number of cases written was 8
```

Options

MAKECASE has the following options:

- Sort the data file by case IDs and record IDs
- Check the sequence of the case IDs. MAKECASE will send an error to the CASE.ERR file each time it encounters a case ID that is not one greater than the previous case ID.

2.3 MERGE

What it does

MERGE combines data from matching cases in two different data files. In the output file, data can be appended to the end of the case or moved toa different location.

You can use MERGE to select cases from one data file without moving any data from the other file (see *OPTIONS* below). MERGE has versatile command language that is easy to use.

When to use it

Use MERGE whenever you want to combine data from two files on a case-by-case basis.

MERGE is effective if you have data collected from a questionnaire and need to match it to the information contained in a "demo deck." You can also use MERGE to combine cases if you enter the data from category questions in one data file first and then code the open-end questions in a different data file later.

How to use it

At the operating system prompt, enter:

```
MERGE
```

MERGE takes cases from an *input* file*,* appends data for each matching case from an import file and writes the combined cases to an output file. You can append the entire matched case from the import file to the end of each corresponding case from the input file, or you can move the entire case or parts of the case to a new location in the output file.

At a minmum, you must provide the file names and types of both input and import files, an output file type, the case length for the output file, and case ID location for both files. If the CFMCDATA environment variable is set, MERGE will look for a TR file there. If the CFMCDATA environment variable is not set, MERGE will look in the current directory. If you need to specify another directory (or another file type), press return and enter the full path and data file name.

Input files

MERGE accepts the following file types:

- ASCII
- Binary
- Swapped Binary
- CfMC System (TR)
- CfMC Phone File (FON from Survent)

Output files

MERGE can create the following types of files:

- ASCII
- Binary
- Swapped Binary
- CfMC System (TR)

The case length must be an even whole number, and large enough to include the length of the input file plus the number of columns you are adding from the import file. The default output length is the length of the input file plus the entire length of the import file. The minimum case length is 80 columns and the maximum is 25,000 columns.

Moving Data

You can move data to a new location in the output file. You must indicate the *from* and *to* data locations. If there is data in the input file in the *to* location, MERGE will write the new data in its place in the output file. You can move up to ten different data locations in one run.

> ***NOTE:*** *Moving data to a new location will write over any existing data. This is not recoverable! Make sure you have a backup copy of your data file before you use MERGE.*

When you have completed the first section of screens, MERGE displays a summary screen and gives you a chance to change options from the defaults. MERGE also gives you the opportunity to save the specs it generates if you want to use them again.

You can enter `terminate` in a text field to quit MERGE when you are entering file information and selecting options. Once processing has begun, you can press Ctrl-Y to terminate the run.

Below are the default file names for the files produced by MERGE. Remember, if you are going to run MERGE repeatedly, you will want to assign unique file names so you do not write over existing files.

Name	Ext.	Description
MRGDT	TR	data file
MERGE	PRT	print file

Writing MERGE specifications in Mentor

All legal input and output options may be used on the ~INPUT and ~OUTPUT statements. You must have two input statements and one output statement to use

Merge. Here is an example merge input/output section:

```
~input  main study=in1 number_input_buffers=2
~input  openends study=in2 new_buffer
~output merged
~merge
```

Within the ~MERGE block the following must be answered:

```
PRIMARY=
PRIMARY_KEY=
SECONDARY=
SECONDARY_KEY=
```

The values for primary and secondary should be set to the study names of two of the input files currently open. Based on the example above, these would be set to in1 and in2 respectively.

The PRIMARY_KEY and SECONDARY_KEY are the fields on which one wishes to match and should be set to a value that evaluates to a string. Here are some examples:

```
PRIMARY_key=[1.4$]
PRIMARY_KEY=SUBSTITUTE([1.4$],"a","0")
PRIMARY_KEY=[1.2$] join [11.2$] join [21.2$]
```

MERGE options

The ~MERGE options are either set to a value, as in <OPTION>=VALUE, or turned on or off via a minus sign, as in option, -<OPTION>. Except for the APPEND_LOCATION option, the setting last seen for any option is the one that will be used.

Status

The status command displays all of the ~MERGE options and their current settings. The following specs will produce a listing of all of the ~MERGE options

available:

```
~input $ study=in1 number_input_buffers=2
~input $ study=in2 newbuffer
~output null
~merge
primary=in1   primary_key=[1.4$]
secondary=in2 secondary_key=[1.4$]
status
~end
```

Here are the results of the status command showing all of the options and their default values:

```
-SORT_PRIMARY
-SORT_SECONDARY
WRITE_MATCHED
WRITE_UNMATCHED_PRIMARY
-WRITE_UNMATCHED_SECONDARY
WRITE_DISALLOWED_DUPlicateS
-PRINT_MATCHED
-PRINT_UNMATCHED_PRIMARY
-PRINT_UNMATCHED_SECONDARY
PRINT_SUMMARY
-PRIMARY_DUPlicateS_ALLOWED
-SECONDARY_DUPlicateS_ALLOWED
DISALLOWED_DUPlicateS=error
DISALLOWED_PRIMARY_DUPlicateS=error
DISALLOWED_SECONDARY_DUPlicateS=error
EXEC_MCOPY_IF_MATCH
-EXEC_MCOPY_IF_UNMATCHED_PRIMARY
EXEC_MCOPY_IF_UNMATCHED_SECONDARY
APPEND_LOCation=0
```

SORT_PRIMARY will cause the primary file to be sorted into a temporary file which will then be used to do the actual merge.

SORT_SECONDARY will cause the secondary file to be sorted into a tem-

porary file which will then be used to do the actual merge.

WRITE_MATCHED means that if the primary and secondary key fields match, a case will be written to the output file.

WRITE_UNMATCHED_PRIMARY will cause a case to be written to the output file when there is no matching record from the secondary file.

WRITE_UNMATCHED_SECONDARY will cause a case to be written to the output file when there is no matching record from the primary file.

WRITE_DISALLOWED_DUPLICATES Usually disallowed duplicates are treated the same as unmatched records. If you are writing unmatched primary or secondary records but don't want to write duplicated records, use the minus form of this option.

PRINT_MATCHED means that when a match occurs, a message will be printed indicating this in either the list file or open print file.

PRINT_UNMATCHED_PRIMARY means that when no matching secondary record exists for a primary record, a message will be printed indicating this in either the list file or open print file.

PRINT_UNMATCHED_SECONDARY means that when no matching primary record exists for a secondary record, a message will be printed indicating this in either the list file or open print file.

PRINT_SUMMARY is used to print summary information about the merge into the list file or open print file. The information printed includes the number of matched and unmatched records, the number of records written, the number of duplicates detected, etc.

PRIMARY_DUPLICATES_ALLOWED is used to allow duplicate match fields to appear in the primary input file, and in the case of a match, not to move the secondary data file on to the next record until all of the primary records having that match key have been used.

If, for example, the primary file contains trailer information and the secondary file contains demographics, one can attach demographic information to each trailer record by allowing primary duplicates.

SECONDARY_DUPLICATES_ALLOWED is used to allow duplicate match fields to appear in the secondary input file, and in the case of a match, not to move the primary data file on to the next record until all of the secondary records having that match key have been used.

NOTE: *Using both PRIMARY_DUPLICATES_ALLOWED and SECONDARY_DUPLICATES_ALLOWED in the same ~merge block will result in an error.*

DISALLOWED_DUPLICATES= may be set to error, warn or ok. When a duplicate record occurs in a file in which duplicate records are not allowed, the

record will NOT be thrown away if the corresponding WRITE_MATCHED_xxxxx option has been set and WRITE_DISALLOWED_DUPLICATES has not been set, regardless of what DISALLOWED_DUPLICATES has been set to.

The default setting for the disallowed duplicates ~merge options is ERROR.

The following two options control the disallowed_dups setting separately for the primary and secondary data files.

All of the ~merge settings work on the last option that is set. The underlying effect of using the original disallowed_dups option should be to set both of the new options to the same value. If either of these two options is used, it should only affect the setting of the file it is intended to affect - primary or secondary.

DISALLOWED_PRIMARY_DUPLICATES= Similar to disallowed_duplicates=, but it only applies to the primary input file. It may be set to error, warn or ok.

'**DISALLOWED_SECONDARY_DUPLICATES=** This is similar to disallowed_duplicates=, but it only applies to the primary input file. It may be set to error, warn or ok.

NOTE: _DUPS is an acceptible syntax when using the options above.

EXEC_MCOPY_IF_MATCH is used to execute the merge_copy commands found in the ~MERGE block if a match occurs.

EXEC_MCOPY_IF_UNMATCHED_PRIMARY is used to execute the merge_copy commands found in the ~MERGE block when there is a primary record that has no corresponding secondary record. Since merge_copy always puts data from the secondary record into the primary record, and because all of the secondary record is blank in the case of an unmatched primary, setting this option causes primary data fields to be blanked when there is no match.

EXEC_MCOPY_IF_UNMATCHED_SECONDARY is used to execute the MERGE_COPY commands found in the ~MERGE block when there is a secondary record that has no corresponding primary record.

APPEND_LOCation=0 until PRIMARY= has been set Once set, the append location defaults to the absolute column number of the WORK_LENGTH of the primary input file plus one.

If you use the APPEND or APPEND_ALL command, appending will begin at APPEND_LOCATION. The value of the append location is dynamic. After each append operation the append location is increased by the number of columns appended. The append location may be set to any column within the output length

and it may be reset in the course of a ~MERGE block.

Data Manipulation Statements

MERGE_COPY

The syntax of the MERGE_COPY command is:

```
MERGE_COPY TARGET_LOC_IN_PRIMARY = source_loc_from_secondary
```

The MERGE_COPY command operates essentially the same way that ~CLEAN copy operates, with the exception of several features designed to help make merging data easier.

The biggest difference between MERGE_COPY and COPY is that the MERGE_COPY command locations specified on the left side of the equal sign are automatically taken to be from the primary data file, and those on the right side are taken to be from the secondary. Thus, what would be written in a ~CLEAN block as:

```
"copy in1![41.10]=in2![1.10]"
```

can be written in a ~MERGE block as:

```
"mcopy [41.10]=[1.10]".
```

NOTE: So far, MERGE_COPY does not work with text questions.

APPEND

The append commands syntax is:

```
APPEND_SOURCE_LOC_from_secondary
```

When the append command is used, the data location specified is taken to be from the secondary data file, and this data is appended to the primary data file starting at the location specified by APPEND_LOCATION (see APPEND_LOCATION= above). The value of APPEND_LOCATION changes after each append command so that successive append commands attach the data starting at the latest LAST_USED + 1 column.

APPEND_ALL

This command attaches the entire secondary record to the primary data file beginning at APPEND_LOCATION (see APPEND_LOCATION= above).

Note that in order to append data, the output file length must be sufficiently greater than the TOTAL_LENGTH of the primary data file to accommodate the data being appended. The following two sets of specs will produce the same

results:

First using append:

```
~input mer3a.asc ascii=40 number_input_buffers=2 study=in1
~input mer3b.asc ascii=40 new_buffer study=in2
~output mer3c.asc ascii length=80  '' Make the output longer.
~merge
primary=in1         secondary=in2
primary_key=[1.4$] secondary_key=[1.4$]
append [1.40] '' Default append_loc is primary worklen + 1
```

And then using MERGE_COPY:

```
'' Extend the work_length on the primary.
~input mer3a.asc ascii=40 worklen=80 number_input_buffers=2
study=in1
~input mer3b.asc ascii=40 new_buffer study=in2
~output mer3c.asc ascii
~merge
primary=in1         secondary=in2
primary_key=[1.4$] secondary_key=[1.4$]
merge_copy [41.40]=[1.40]
```

Proc=

You can define a proc and use it in ~MERGE. The commands used in the proc should be those you would use in a ~CLEAN block (i.e. not ~MERGE commands).

MERGE_defaults

The ~SET_OPTION_MERGE_DEFAULTS= allows the user to define a string of merge commands that will be executed each time a ~MERGE block begins.

The string that you set MERGE_DEFAULTS equal to is not cracked until a ~merge command block is started. The string may contain any command that would be legal inside a ~MERGE block including meta commands. The string may be continued by using "&&". Backslash-Ns that appear in the string are recognized as a new line character. Note how the use of \n in the default string below

allows one to use the >QUIT command as part of the defaults setting.

```
merge_defaults=">quit errors=1\n primary_dups_allowed" &&
                "primary_key=[1.5$] secondary_key=[1.5$]"
```

EMPTY_CASE

EMPTY_CASE is a special variable that only is used from within procs called by a ~merge block. It should always be prefaced with a study name, as in 11EMPTY_CASE. EMPTY_CASE is true when it is inside a ~MERGE block and the record referenced by "studyname" is the nonexistent side of an unmatched pair.

Sample output

The MRGDT.TR file contains the combined cases. The MERGE.PRT file contains a list of errors for missing and bad cases, warnings when data is being moved to columns that already contain data, and a summary report which includes counts of the number of records that matched and were written to the output file. Below is sample of MERGE.PRT. In this example, data was moved to a location that already had data in it (columns 25-28), and two cases did not match.

```
**** WARNING CASE 0001 IS NON-BLANK IN RECEIVING FIELD: 25.4 ****
**** WARNING CASE 0002 IS NON-BLANK IN RECEIVING FIELD: 25.4 ****
INPUT CASE 0003 WITH MATCH FIELD = "0003" HAS NO MATCH IN IMPORT FILE
**** WARNING CASE 0004 IS NON-BLANK IN RECEIVING FIELD: 25.4 ****
INPUT CASE 0005 WITH MATCH FIELD = "0005" HAS NO MATCH IN IMPORT FILE
**** WARNING CASE 0006 IS NON-BLANK IN RECEIVING FIELD: 25.4 ****
**** WARNING CASE 0007 IS NON-BLANK IN RECEIVING FIELD: 25.4 ****
**** WARNING CASE 0008 IS NON-BLANK IN RECEIVING FIELD: 25.4 ****
SUMMARY
-------
NUMBER OF MATCHED RECORDS: 6
NUMBER OF INPUT RECORDS WITHOUT MATCHES: 2
NUMBER OF RECORDS WRITTEN TO OUTPUT: 8
NUMBER OF RECORDS IN INPUT FILE: 8
File with merge listing is MERGE.PRT
Output data file name is MRGDT.TR
```

Options

Select the options you wish to change by entering the numbers of those options. The first two options, (1) cases to write out and (2) cases to print out require that both files be sorted on their respective match fields (case IDs). You will be prompted to indicate if the files are already in order or need to be sorted. The options to MERGE are:

1 Cases to write out

This option controls which records are written to the MRGDT.TR file; the default is to write out all records from the input file. You can change this to write out:

- all records from both input and import files
- all records from the input file*
- only matching records from the input file*
- only unmatched records from the input file
- all records from the import file
- only unmatched records from the import file
- all unmatched records from both input and import files
- no records (compare only)

* Use these options if you want to copy only certain cases from the input file without moving any data from the import file. This will create a new data file that is a subset of the original input file, with cases that either match or do not match the cases in the import file.

2 Cases to print out

This option controls which records are written to the MERGE.PRT file; the default is to include only unmatched records from the input file. You can change this to include:

- all unmatched records from both input and import files
- unmatched records from the input file
- only unmatched records from the import file
- all records from both input and import files
- all records from the input file
- only matching records from the input file
- all records from the import file
- no records

3 Duplicate handling

The default is to use all duplicate cases from the input file, and take data from only the first duplicate in the import file. You can change this to use only the first

duplicate from the input file and append data from the first duplicate in the import file.

4 Duplicate messages

This option controls when MERGE generates messages about duplicate cases. These messages appear in the MERGE.PRT file. The default is to ignore duplicate cases in the input file, and complain about duplicate cases in the import file. You can have MERGE complain about duplicate cases in the input file only, duplicate cases in both files, or not create any messages about duplicates.

5 Data Overwrite messages

This option controls the number of data cases MERGE will complain about when it is moving data to a location where data already exists. These messages appear in the MERGE.PRT file. The default is to create messages for the first 10 cases. The range you can choose from is 0 (no messages) to 99.

2.4 DBUTIL

What it does

DBUTIL lets you look at items in a DB (database) file or copy items from one DB file to another DB file.

When to use it

DBUTIL lets you do housekeeping on a DB file. You can copy DB items from one file to another, or, in the case of duplicate items, you can select which records you wish to make active.

You should be familiar with basic DB commands in order to use DBUTIL. See *Appendix A: Meta Commands* for details on >USE_DB and >CREATE_DB.

How to use it

At the operating system prompt, enter:

```
DBUTIL
```

You will be prompted for a Spec File and a List File; both are optional. (See *Mentor Appendix D: CfMC Conventions* for a description of Spec Files and List Files.)

From the DBUTIL prompt, you can choose Copy, Reveal, Help(?), or Quit.

Note: Many of the DBUTIL commands are meta commands, and require the meta symbol (>) to be recognized. So, when a meta (>) is included in an example, be sure to include it when you enter the command.

Options

Copy

1. **To copy a DB file, you must open it first. At the DBUTIL prompt, enter:**

```
>USE_DB file1
```

2. **To copy the items to another DB file, you must open it in ReadWrite mode. Enter:**

```
>USE_DB file2,READ_WRITE
```

3. **Once both the DB files are open, copy the items. Enter:**

```
COPY ALL
```

Reveal

1. **Open the DB file in ReadWrite mode. Enter:**

```
>USE_DB filename,READ_WRITE
```

2. **To show specific items from the DB file, enter:**

```
REVEAL
```

3. **Enter the name(s) of the item(s) you want to see. You can use an ampersand to continue a list of items on the next line. For example:**

```
Name(s) of item(s) to deal with->AGE,INCOME,GENDER,&
OCCUP,RATING1
```

All records for each selected item are displayed. In the case of multiple entries for a single item, the record with an asterisk on the line is currently active. Enter the line number of another record if you wish to make it the active record. In the example below, the

second item is currently being used by the database.

```
-->>USE_DB MYDB3,READ_WRITE
-->REVEAL
Name(s) of item(s) to deal with -->AGE
Look for all versions of DB item AGE
<-------When item stored-------> Type Length
1: WED JUN 23 1995 09:09  1 86
2: * THU JUN 24 1995 13:20 1 98
We have found 2 items. Which one to make ACTIVE? -->
```

4 **Or, for a full list of the items in the DB file, enter:**

```
>USE_DB filename,READ_WRITE
>LIST_DB filename
```

Note: *When DBUTIL has completed an option (COPY or REVEAL), all DB files will be closed. You must open the database file again to perform another operation.*

2.5 RAWCOPY

What it does

RAWCOPY recovers corrupted CfMC data files (TR). RAWCOPY looks at each data base, tries to recover it, and discards the cases that are not valid.

When to use it

Use RAWCOPY with data files that do not open. Possible causes of file corruption include system crashes while the data file is open, bad sectors on hard disks or diskettes, errors in file transfers, or files that have been altered, possibly with a debug utility.

How to use it

At the operating system prompt, enter:

```
RAWCOPY
```

At a minimum, you must provide the data file name, and an output file name (if you enter a 1-to-8 character-long name, RAWCOPY will add the TR extension for

you, if you enter the complete path with the file name, you need to include the TR extension).

Input files

RAWCOPY only works on CfMC data files.

Output files

RAWCOPY creates a data file containing all the cases it could recover.

Options

You can have RAWCOPY check for a specific case ID length; the cases that don't have the case ID length you specify will be dropped. If you do not use this check, you run the risk of recovering previously deleted cases. If you choose to not use this check, enter "0" when you are prompted for the number of digits in the case ID.

You can have RAWCOPY fix up cases that have had text pointers corrupted or deleted. You must enter the starting location of the text area. RAWCOPY will write the "back pointer" back to the "front pointer" position for all text in the text area that does not currently have a front pointer pointing to it. If you do not wish to recover text pointers, when you are prompted for a 'textstart' location. (See the *Survent* manual discussion of TEXT questions for more information.)

2.6 REFORMAT

What it does

REFORMAT takes a data file and a set of CfMC variables from a CfMC QFF file or DB file and produces an ASCII file of fixed or delimited format with only the variables you requested. REFORMAT also creates a data map for the data file.

You can use a compiled Survent questionnaire, or specific variables from a Survent questionnaire, use Mentor variables or tabsets, or create new variables using data location references. If you use existing variables, you can get all variables in a range, variables of certain types (eg. CAT or TABSETs) or specific variables.

Coded items can be spread as response codes or in 0/1 format. You can include long open-ended (TEXT) questions in the spread data or just standard data points.

When to use it

Use it when the end user needs data in an ascii format. REFORMAT allows you to use a compiled QFF file if you are a Survent client (with some additional controls). If you are a Survent OR Mentor client, you may use variables from a DB file rather than the questionnaire file. The main advantage of using a DB file for Survent clients is that you can get a subset of the variables from a questionnaire without putting special controls in the questionnaire or recompiling a questionnaire.

How to use it

At the operating system prompt, enter:

```
REFORMAT
```

At a minimum, you must provide the data file name (if you enter a 1-8 character-long name, REFORMAT will add the TR extension for you, if you enter the complete path with the file name, you need to include the TR extension), a name for output files, and the size for text questions in the reformatted data (10-5000 characters). Use a unique output name. If a file with the same name already exists, REFORMAT will rename the existing file.

Input files

REFORMAT accepts the following file types:

- ASCII
- Binary
- Swapped Binary
- CfMC System (TR)
- CfMC Phone File (FON from Survent)

Output files

REFORMAT can create the following files: a data file (RFT), a map file (RFL) and a data definition file (DEF). You can create data definitions that match the spread data for SSS-XML, SPSS, SAS, SQL “readable” files, QUANTUM, and UNCLE, or CfMC packages COSI, Survent, or Mentor.

Sample output

The data file (RFT)

The three types of data records you can export are 1) fixed format, 2) delimited, or 3) card-image. By default the data is spread in fixed format. See OPTIONS to change the output from fixed format. This is a sample data file for a multiple-response CAT question spread as codes. (See the map file section for details about this data file.)

```
00017
0002
00037
0004
0005456
0006
0007
0008
0009126
001024
```

The map file (RFL)

The map file (RFL extension) lists information about the question data, recode values, questions and responses, and additional information depending on the question type.

This is a sample map file for a multiple-response CAT question spread as codes.

(This is question 6a from the Roadrunner questionnaire.)

```
rrunr.rfl: Old record length=640 New record length=10 Page 1
Q= Label Type FromLocation ToLocation RefmtType MaxResp
The case ID will be in columns 1.4
Q QN6A CAT [31] --> [5.6] Type=CODE Max=6
X 1st=[5] 2nd=[6] 3rd=[7] 4th=[8]
X 5th=[9] 6th=[10]
T Q6a. Which entertainment was participated in during the past three
months?
R 1 31^1 --> X=1 Video games
R 2 31^2 --> X=2 Billiards
R 3 31^3 --> X=3 Fun House
R 4 31^4 --> X=4 Musical Revue
R 5 31^5 --> X=5 Dunk the Moose
R 6 31^6 --> X=6 Other
R 7 31^7 --> X=7 Don't know/refused, use Mentor
```

The record types printed are identified by the letter in column one:

Letter	Record type
Q	Question description information
X	Extra line for some CATEGORY, NUMERIC, EXPRESSION and SPC questions
T	Text line
R	Response item for CAT and FLD questions

Q line question descriptions include:

Q-line question	Description
Q=label	The question label or number (e.g., CAT01)
Type	Question type (e.g., CAT for CATEGORY)

Q-line question	Description
FromLoc	Data location and width in the original data (e.g., [1/29])
ToLoc	The data's new location in the RFT file (e.g., --> [33.8])
RefmtType	The reformat type varies according to the question type
	For CATEGORY questions:
	Type=1/0 (default, see data file option 7)
	Type=CODE (see data file option 6)
	Type=MOVE (see data file option 5)
	For FIELD questions:
	Type=MFLD for multiple-response FLD questions
	Type= is not specified for single response FLD questions
	For LOOP questions:
	Type=UNWIND (if making DEF files)
	Type=SAVE (if not making DEF files)
MaxResp	The number of separate responses allowed (e.g., Max=1)
LotusItem	For comma-delimited format only, this is the LOTUS column number for this data (e.g., Item#=6)

X line extra descriptions include:

- The column grid that determines the data location in the RFT file for multiple-response CATEGORY questions spread in either response code or punch format.
- For FIELD or coded ascii questions, the X line appears only for multiple-response FIELD questions in the same format as multiple-response CATEGORY questions spread as responses {!RFT_CAT_RESPONSE}).

The X line indicates the columns in the RFT file for each of the possible responses. In this example there are 10 possible responses and each response uses four columns for a total of 40 columns:

```
Q RADIO FLD [1/7.4] --> [7.40] Type=MFLD Max=10
X 1st=[7.4] 2nd=[11.4] 3rd=[15.4] 4th=[19.4]
X 5th=[23.4] 6th=[27.4] 7th=[31.4] 8th=[35.4]
X 9th=[39.4] 10th=[43.4]
T This is the map for a multi-response FLD question
R KDFC
R KQED
R KVAL
R KJAZ
R KALW
R KFOG
R KAFE
R KCAF
R KBBF
R KCDS
R KLVM
R KIQI
R KMEL
R KPLS
R KREO
```

- For NUMERIC questions the X line indicates the valid range of responses and any allowed exception codes. It would look like this:

```
X Range=1-5000  Exceptions=DK,NA
```

T lines contain the question text.

R line recode descriptions (CATEGORY or FIELD) include:

- Response code
- The original data location and punch (for CATEGORY questions)
- Location and punch or response code in the spread data file response text (for CAT questions)
- ToLocation for CATEGORY (punch) variables vary for the reformat type:

For MULTI_CAT_01 option, you will see the ToLocation and punch in the spread data file.

For option 5 in the REFORMAT program you will see X.width and the response code, where X refers to the spread data column locations on the X line.

```
X 1st=[32.2]  2nd=[34.2] 3rd=[36.2]
R 03 1/30^3 --> X.2=03 YELLOW
```

In this example, if 03 was the first response for this question then the code 03 would be in columns 32 and 33 in the spread data file.

Qff or Db file

After asking the name of the data file to reformat, the next screen will ask whether you will be using a QFF file for the reformat or a DB file. Survent users will want to use the QFF file for a complete dataset and/or controls based on the questionnaire coding. Mentor users may only use the DB file and will specify which variables they want. Survent users may use the DB file especially if they only want a few variables in the output.

Map and Data File Options

REFORMAT lets you change what information is included in the map file and how it is formatted (MAP file options screen) and what information appears in the data file (options screen).

Map File Options

The Map file contains information about the variables in the data file. Here is a description of the options:

1 Exclude “from” columns

By default, the map file includes information about where the data came from and where it was spread to in the new data file. If you want to exclude the “from"” columns, use this option.

2 Exclude page headings

By default the program provides page headings with the name of the study, the page number, and column headings for the information listed for each question. If you wish to exclude this information, choose this option. You can use this option when you are generating a map file to be read by another software program.

3 Exclude page breaks

If you do not wish to have page breaks in the map file, choose this option. This would be useful if you had a printer that could not properly print the pages with page breaks.

4 Change page length

Use this option to change the page length from the standard 66 lines per page.

5 Change page width

Use this option to change the page width. By default the width is unlimited.

6 Change column headings (delimited files only)

By default, the program prints the variable names at the top of each column in delimited files. Choose this option to change to "A"-"Z" type headings like those used by many spreadsheets.

7 Use “SAMEAS” for duplicate code lists

The map file includes a complete code list for every code list question by default. If you choose this option, for each time you have the "SAMEAS" command in the questionnaire, REFORMAT will write one line with the notation "This uses the same code list as question XXX."

Data File Format Options

You are presented with the following data options with REFORMAT:

```
Enter the Options you want, or press <ENTER> only to make a FIXED FORMAT
ASCII data file with no variable definitions.

DATA FORMATS:

1) FIXED format ASCII data w/ optional data definitions for CfMC
Mentor/Survent/COSI, or SAS, SPSS, QUANTIME, UNCLE

2) DELIMITED format (comma or tab delimited for spreadsheets, etc.)

3) CARD-IMAGE format (80 column records with Case and Card IDs)
```

1 Write fixed format with data definitions (Mentor, COSI, SPSS)

Fixed format ASCII data spreads the data in the same relative position for every question across respondents. Records will be as long as necessary for the data spread. The data definition file(s) created by REFORMAT contain variable descriptions for different data processing packages. Generally, this includes the question text, new location, type, and code list where applicable. In addition to variable definitions, some data descriptions include automatic table creating specifications (Mentor, QUANTIME).

The options presented for types of data definition files and their file extensions are:

a) None

b) CfMC Survent
Questionnaire specifications matching the spread output (QSP)

c) CfMC Mentor
Mentor tabsets matching the spread data (DEF)

d) COSI
Variable definitions export to CfMC companion Windows-based tabulation/printing/charting package, file extension (DEF)

e) SPSS
Definitions to load into this data processing package (SPS)

f) SAS
Definitions to load into this data processing package (SAS)

g) QUANTUM
A tabulation package (QUA)

h) UNCLE
A tabulation package (UNC)

i) SQL "readable" files

j) TRIPLE-S XML
A market research data definition standard (SSS)

The RFT file exported using these options is created with all response list items spread as response codes by default (except that SAS multi-response CAT questions are spread as 0/1 variables).

Data files created with data definition specs other than SAS will only do 0/1 variables if you use the “make 0/1” variables reformat option (MULTI_CAT_01).

LOOP variables are spread in such a way that there is a separate iteration for each item answered in code list order.

For COSI, SAS, SPSS, TRIPLE-S XML and UNCLE spec file generation, the variables created for other packages generate new names to create separate variables in cases where there is a multi-response question or there are questions inside of a loop. In order to create unique variable names with these multiple iterations, REFORMAT truncates variable names as many characters as necessary to add an extension.

For multiple-response CATEGORY questions, REFORMAT adds the numbers "01" through "99" to the end of the variable name; if the name is greater than six characters it is truncated by one or two characters; if there are more than 99 responses allowed, REFORMAT uses the ASCII lettering sequence starting at "AA" to "ZZ."

For LOOPS, REFORMAT adds letters "A" through "Z" to the end of the variable names less than eight characters; if the name is eight characters long, the last character is replaced by an "A" to "Z".

For multi-response questions within loops, REFORMAT adds the extension “01A” for the first name; names greater than five characters will be truncated as necessary. For example, a question named "HOSPITAL" which has 10 responses and 5 loop iterations will have its iterations named "HOSPI01A" through "HOSPI10E". When names are truncated and the extension added, it is possible to end up with duplicate names; if this occurs, REFORMAT generates a warning message.

Note that other software packages, although supported in general, may not match the types of variables you wish to export for those packages. If this is a problem, you can use the RFL (MAP) file to generate a coding language that will work with

your other software as a basis to create your own variables.

2 Write delimited format (for spreadsheets)

Delimited data is spread with a standard width for each question, and a delimiter of your choice between variables. Common delimiters used are tab, comma and "blank." Note that this also creates fixed format data (it is in the same relative position across respondents). Quotes are places around variables with "string" data; numeric data is recorded without quotes around it. Data from the following question types will be enclosed in quotes: SPC subtypes 3, 4, 7, 9, and V; PHONE subtype G; and VAR. Comma-delimited format adds from one to three columns to the original data width (one comma plus quotes around data).

In addition, Reformat has a new "**MAXIMUM_DELIMITED_FIELDS**=###" option. This controls how many items to put in each output delimited file. You can now have reformat create more than one delimited file with each file having a specified number of fields. This is so that users that have databases with a maximum number of supported fields can read each of the files separately into the program; for instance, EXCEL has a maximum of 255 fields in a file. If you use this feature, REFORMAT will make as many files as necessary to create the output, with each file having a maximum of ### fields.

The minimum value is 10 fields, the maximum is 32000. If you have a question variable translating to more than the maximum_fields value, the program will generate an error and request that you break the question variable into smaller pieces and try again.

If a question variable has more fields than are available to fill the current file, all of the fields of that variable will be put in the next file, etc. The files are named as <study>_#, where # is 1-999. The first file is <study>_1, then <study>_2, etc. until all variables are placed in some file.

An exception to this feature is the !LOOP variable; if REFORMAT finds a !LOOP variable in the file, it will stop the run and deliver an error message.

3 Write card image format

Card-image data generates a case ID and record ID for each 80-column record generated, and then the data that fits on that record. REFORMAT numbers card IDs sequentially from 01 to nn for the number of records created for each respondent. The same number of records are written for each respondent regardless of the amount of data in their particular data record. REFORMAT writes record descriptions in the MAP file in record number/column format (e.g., 2/23 means card number 2, column 23, or absolute position 103).

Data File Spreading Options

Next you will see the following menu:

SPREAD OPTIONS/OTHER ISSUES:

0) Eliminate non-labeled questions (eg. rotate seeds)

1) Convert NUMERIC alpha exceptions to numbers
(eg. 01-10,,dk becomes 001-010,,999)

2) Convert CAT/FLD codes to sequential numbered codes (01-99)

3) Convert CAT/FLD codes to text ("yes"/"no" instead of 1/2)

4) Convert CAT/FLD codes to 0/1 variables ("135" to "10101")

5) Expand binary only (CAT/TEXT), keep FLD/NUM/VAR data as is

6) Expand multi-response CATS ONLY when writing CAT/TEXT

0) Eliminate nonlabeled questions

This removes any questions from a questionnaire with “temporary” labels, the idea being that if you didn’t label them you probably don’t need to save them.

1) Convert NUMERIC alpha exceptions to numbers

This changes alphanumeric exception codes to numeric codes in case your data needs to be all numeric format. If you have a numeric variable with exceptions such as “DK” for don’t know, it will convert that to a number higher than the range allowed for the numeric values. So if the range allowed is 1-99, it makes the exceptions 999,998,etc. backwards to 997. This will make the variable wider by one charcter.

2) Convert CATEGORY/FIELD codes to sequential numbered codes

This makes all the codes start at “1” for 9 or fewer codes, “01” for 10 to 99 codes, and “001” for 100-999 codes, and numbers all the codes sequentially from that point when converting the data. This is another feature designed to change possibly alphanumeric or out of sequence codes to seqential numeric.

3) Convert CATEGORY/FIELD codes to TEXT

This writes out the text of the response instead of the response code. It is especially useful when creating delimited files to be read into Access, Excel, or other

databases.

4) Spread CATEGORY/FIELD questions as 0/1 variables, not codes

This writes code variables as 0/1 variables, meaning each code will generate a 0 if it is not picked as a response, and a 1 if it was, creating strings of 0 and 1 codes. If you choose this option, you will be asked whether to convert single response CATEGORY questions, multi-response CATEGORY questions, single response FIELD questions, and/or multiple response FIELD questions.

5) Expand binary only (CAT/TEXT), keep FLD/NUM/VAR data as is

For Fixed format datasets only, appends CAT and TEXT question data (in the whatever format specified) at the end of the dataset, leaving all FIELD, NUMERIC, and VARIABLE data in its original location.

6) Expand multi-response CATS ONLY when writing CAT/TEXT

This is a sub-option of option 5, it says to only spread multi-response CATEGORY type questions (and TEXT questions) at the end of the dataset, NOT single-response items.

You are also asked whether to include TEXT questions in the dataset output. If your output is fixed format, you are asked the maximum length of a text item. If your output is card image format, it only allows a maximum of 70 characters per item to stay within the 80 column card image requirements.

Troubleshooting

Possible reasons for the program not to run would be a DB file and TR file that do not match, or if you used TABSET definitions that were complex variables. The program will write out data even if it finds errors when matching the data to the variables, so it is your responsibility to check the integrity of the data file you create. You should check your data file carefully if you see a warning messages like this:

```
(WARN #891) There were too many responses found in the data here!
Error on question 0.10 and case ID 0001
```

If your files are very large, you may get core size errors. See data file option #8 (above) to increase maximum core size.

REFORMAT exports "SQL readable" files

The ~reformat option "sql_data=(databasename=<database name> table-name=<table name>)" will create a .def file with an SQL table definition, a .tab file with SQL insert statements and a matching data (.rft) file to be uploaded to the table in an SQL database.

Example:

```
~input mt3710 ;
~qfffile mt3710
~specfile mt3710a
~reformat sql_data=(databasename=cfmctest tablename=mt3710a)
~end
```

Options include:

databasename - name of the data base

If specified, the .def file will use the name that you provided and will begin with these two lines:

```
create database if not exists mydatabasenamehere;
use mydatabasenamehere;
```

If not specified, the .def file will begin with the above two lines commented out.

tablename - name of the table (required)

include_size_on_text - using the minus form removes the size spec from text variables in the .def file (e.g. gives you text instead of text(2000)). This option exists because not all data base programs allow the size spec in their syntax. The default is "on."

num_exceptions_separate - causes NUM questions that have alpha exceptions to be saved as two variables, one with only numbers and the other with only the exception codes. the variable for the exceptions will have _extra appended to its name, as in Q1 and Q1_extra

- without this option NUM questions with alpha in them will be type char
- with this turned on you will get a type numeric and a type char variable
- the default is "off"

Here is an example .def file:

```
create database if not exists bank;
```

```
use bank;
create table if not exists mike (
caseid char(4),
                    havecard char(1),
                    cardtype_1 char(1),
                    cardtype_2 char(1),
                    cardtype_3 char(1),
                    cardtype_4 char(1),
                    cardtype_5 char(1),
                    cardtype_6 char(1),
```

The .rft file will be a tab-delimited version of the data that can be added to a database using the “LOAD” command.

The CfMC variable types are converted to SQL types as follows:

CfMC Type	Sql Type
FLD	char
NUM (without decimal)	numeric
NUM (with decimal)	dec
EXPR (without decimal)	numeric
EXPR (with decimal)	dec
VAR	char
TEXT	text

The default for all other variable types is “char” (!spc, !phone, etc.).
If there are more than 100 characters of text, they are concatenated using concat ("first 100 chars of text","next 100 chars","more text"). If less than 101 characters, then just do ‘here is some text’. Enclosed quotes (") are converted to \" and ' to \'.

Using spec language

You can use Mentor's ~REFORMAT commands to execute REFORMAT commands in batch mode. Use the "S" option on the final REFORMAT screen to save a spec file to reproduce your run or write your own specifications. Below are a list of commands you could use when writing or modifying a spec file.

The following commands are required in your spec file:

~INPUT <filename> <filetype>	Name of the data file to process
>USEDB <filename>	Use existing variables from DB file
~QFFFILE <filename>	Use a QFF file for the reformat
~REFORMAT	Invokes the default reformat options
~END	Ends the program

To generate lists of variables by type or name:

>LISTDB,listv^dcl,&	List the items to file "listv^dcl"
TYPE=VAR=14567,	Use types VAR, FLD, NUM, CAT, or TEX respectively
TYPE=TABSET,	type TABSET (Mentor only)
SORT=qqnum(q1-q23),	Use range of qs from q1 to q23 (Survent only)
PATTERN=(qn2*,T*,xyz),	Use qs with names starting with T or qn2 or xyz
TEMPLATE="!"	Write each item out to the file on a separate line

The following are ~REFORMAT options, placed after the ~REFORMAT command and before the ~END command:

1 **Use variables option:**

Variables=(var1 var2,var3	Enter items by name or make a list
&list	Read in the list of items to spread

var4 var5 var6)	Items can be on separate lines, spaces or separated with commas

2 File type options:

DELIMITER=COMMA/TAB/SPACE/<char>	Write delimited file with delimiter of <char>
DELIMIT_NAME_FIRST	Write name of variable on first line of file
CARD_IMAGE	Write card image file (80 byte records)

3 Miscellaneous options:

DO_WHAT_YOU_CAN	If there are data errors, attempt to make file anyway
-USE_CFM_LABELS	Ignore unlabelled/temporary label questions (CFM######)

4 Map file options:

-MAP_FILE	Don't make map file
MAPFILE=(	Start list of map file options
-FROM_FILE_INFO	Don't include data locations data came FROM, only TO
-HEADERS	Don't include page headings
-FORM_FEED	Don't include form feed at page breaks
PAGE_LENGTH=###	Change page length from 66 lines per page to ###
EXCEL_NAMES	Use "AA" - "ZZ" type variable names in delimited files instead of real names

SAMEAS_IN_MAP	Write "SAMEAS list at Q2a" instead of remaking list, if list uses "SAMEAS" feature in Survent.
)	End of map file options

5 Category/Field variable options:

MULTI_CAT_01	Write 0/1 codes (and matching variables) for multi-response category questions
APPEND_CAT_DATA	Leave other data alone, spread category data at the end
-EXPAND_SINGLE_CATS	Don't leave single-response category questions, leave as punches
RENUMBER_RESPONSES	Change response codes to sequential numeric codes
SINGLE_CAT_RESPONSE	Expand single-category questions as response codes (override 0/1 coding)
USE_RESPONSE_TEXT=####	Exports the text of the response item instead of the response code

6 Other question options:

RECODE_ALPHA_EXCEPTIONS	Change an alphanumeric values to high numeric values in NUMERIC questions.
TEXT_AS_VAR=###	Expand TEX questions in standard text area using length of ###

7 Data definition file (DEF) options:

SURVENT_SPECS	Generate Survent QSP file matching spread data
HARDCOPY	Associated Survent file

CHK	Associated Survent file
SUM	Associated Survent file
MENTOR	Mentor table-building DEF file matching spread data
CLN	Cleaning specifications for Mentor
COSI_SPECS	Specs for COSI
SPSS_SPECS	Specs for SPSS (SPS)
SAS_SPECS	Specs for SAS (SAS)
QUANTUM_SPECS	Specs for QUANTUM (QUA)
UNCLE_SPECS	Specs for Uncle (UNC)
SSS_XML	Specs for Triple-S XML (SSS)

Chapter 3
DATA ANALYSIS

3.1 HOLE

What it does

HOLE produces a report called a holecount (also known as a marginal). A holecount is a display of the counts of the punches in the data for each column. Data used to be recorded by machines that punched holes into cards and this is where the idea of having holes (often called punches) to represent data orginated. The possible punches in one column are 1 to 9, 0, X and Y.

HOLE creates simple counts and percentages which are easy to read. Columns are labeled with the record number and column number, or can be labeled with the absolute column number. Percentages are based on the total number of cases. HOLE will produce a report for a maximum of 2,000 columns per run.

When to use it

A holecount report allows you to view data from an entire data file in a few pages. Use HOLE to ensure that numbers in cells in a table match those in the data. You can also use HOLE after you have transferred a data file across platforms to verify that it has the same counts for each punch. You can also use holecounts to determine how to group response sets for a stub (SCAN works for this as well).

How to use it

At the operating system prompt, enter:

```
HOLE
```

Follow the instructions for each screen. At a minimum, you will have to supply a data file name (and accept all the defaults). You can use multiple input files, as

long as they are the same file type. Use commas to separate file names, or wild cards ("?" for letters, "#" for numbers, or "*" for anything) to select a group of files.

You can press **ESC** to exit most screens, or enter `terminate` in a text field to quit HOLE.

When you have completed all the screens, HOLE displays a summary screen which indicates the options you have selected and gives you a chance to change them before you run the report.

Input files

HOLE accepts the following file types:

- CfMC System (TR)
- ASCII
- ASCII Card-image (multiple records per case)
- Binary
- 'Swapped' Binary (from other computer types)
- CfMC HP3000 SPL format (DTA)
- CfMC Phone File (FON file from Survent)

Output files

The default filename for a holecount is the data file name with an extension of HOL (for example “rrunr.HOL”). Remember that if you are going to run HOLE repeatedly, you will want to assign unique file names so you do not write over existing files.

Sample output

A HOLEcount report has the following headings by default:

Heading	Description
Data column:	Case/Column number
Total:	The number of cases in the data file. Percentages are based off of this number.

Heading	Description
No Answer:	The number of cases that had no responses in the column.
Multipunch:	The number of cases that had more than one answer in the column.
Any Answer:	The number of records that had any answer in the column. ("Total" minus "No Answer" = "Any Answer")
Total Answers:	The total number of responses in a column. This number will be larger than "Any Answer" if there are any multipunches in that column.
1-9,0,X,Y	Total number of responses for each column.

On the following page is a sample of the report produced by HOLE for the first ten columns of data from the Roadrunner data file.

Computers for Marketing Corp. - Holecount of rrunr.tr - Page 1

Columns 1 - 10

Data Column	Total	No Answer	Multi -Punch	Any Answer	Total Answrs	1	2	3	4	5	6	7	8	9	0
1/1	500 100.0%	-	-	500 100.0%	500 100.0%	-	-	-	-	-	-	-	-	-	500 100.0%
1/2	500 100.0%	-	-	500 100.0%	500 100.0%	100 20.0%	100 20.0%	100 20.0%	100 20.0%	1 0.2%	-	-	-	-	99 19.8%
1/3	500 100.0%	-	-	500 100.0%	500 100.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%
1/4	500 100.0%	-	-	500 100.0%	500 100.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%	50 10.0%
1/5	500 100.0%	500 100.0%	-	-	-	-	-	-	-	-	-	-	-	-	-
1/6	500 100.0%	-	-	500 100.0%	500 100.0%	83 16.6%	84 16.8%	87 17.4%	77 15.4%	71 14.2%	-	-	-	-	98 19.6%
1/7	500 100.0%	-	-	500 100.0%	500 100.0%	85 17.0%	79 15.8%	81 16.2%	82 16.4%	93 18.6%	-	-	-	-	80 16.0%
1/8	500 100.0%	-	-	500 100.0%	500 100.0%	91 18.2%	86 17.2%	81 16.2%	79 15.8%	74 14.8%	-	-	-	-	89 17.8%
1/9	500 100.0%	-	-	500 100.0%	500 100.0%	76 15.2%	89 17.8%	88 17.6%	83 16.6%	87 17.4%	-	-	-	-	77 15.4%
1/10	500 100.0%	-	-	500 100.0%	500 100.0%	88 17.6%	75 15.0%	73 14.6%	81 16.2%	89 17.8%	-	-	-	-	94 18.8%

Options

The default is a holecount of all columns, percentaged off the total number of cases. To change the defaults, choose one or more of the following options:

1 Make your own HEADER

Create a title or heading of the holecount report.

2 Print the DENSITY of the columns.

Include a count of the number of multipunches for each column. This shows the number of answers from each record. Density appears as a number under the frequency and percentage.

3 BASE the hole counts.

Use only certain records from the data. You can use a data location to determine which records to use. An example would be a base of [5^1], which means only include those cases that have a one punch in column five. You can also use variables from previous Mentor runs. Enter the name of the DB file in which the variables are defined. Refer to variables by question number or name using the following syntax:

Category (CAT) or Field (FLD) variables	
GENDER(M)	gender category M
AGE(1-5)	ages categories one through five
STATE(AL-CA,NY)	state categories AL through CA or NY

Variable(VAR) or Text(TEX) questions

```
OTHRCARD="CHEVRON"
```

Numeric(NUM) questions

```
AGE>=35
```

Data Location References		
Category	**Numeric**	**Field**
[AGE^1-5]	[STATE#AL,AZ,CA,NY]	[10.5] > 5

Data Location References		
Category	**Numeric**	**Field**
[GENDER^1]	[AGE#1-10,23-55]	
[1/10^1-5]		

4 WEIGHT the data

Weighting is a way to change the relative value of each case. You would probably only want to weight the data in your holecount report if you are comparing it to data in a weighted table. You can also specify how many decimals of significance to print.

You can use data locations or variables to define the weight, see the examples under the BASE option.

5 Specify the PAGE FORMAT

To change one of the page format options, enter the item number of the item(s) you wish to change. You must enter all the options you wish to change at once; separators between the item numbers are not necessary. For example, entering 13 would allow you to change the page length(1) and change the number of decimals in the percents(3).

Option	Default
1) Page Length	66
2) Data Column Percent Base	total
3) Number of Decimals in Percents	1
4) Print Percent Sign	yes
5) Print Frequencies Only	no
6) Print Percents Only	no
7) Rows of All Zeros Printed	yes
8) Data Location Format	<unknown>
9) Compressed Display	yes

6 Do MULTIPLE COLUMN SETS

Allows you to do holecounts on groups of columns. This is useful to check only certain columns or to avoid printing a group of columns that are blank.

7 Make your own FOOTER

Put a note at the bottom of the holecount report.

3.2 SCAN

What it does

SCAN produces tables using data locations or variables from a CfMC DB file. Different variable types will produce different tables:

CAT (category) or FLD (field): Counts for each category with percentages of the total. The No Answer category is a count of the number of cases that did not fit into a category. The Any Response category is a count of the number of cases with any response. (Any Response + No Answer = Total)

When specifying All Variables or All of Certain Types, it prints the "TOTAL RESPONSES" at the bottom of the table.

NUM (numeric): Counts for Total, No Answer, Any Response, Mean, Standard Deviation, Standard Error, Median, Minimum (value in the field) and Maximum (value in the field).

When specifying All Variables or All of Certain Types, it prints values, plus Mean, Standard Deviation, Standard Error, Median, Minimum, Maximum and sum. Numeric variables also have an option to either get stats only, all values, or all values and stats.

VAR (variable [open-ended]): Counts for Total, No Answer, and Any Response, plus a breakout of all the different values in the field.

TEX (text [special format open-ended]): Counts for Total, No Answer, and Any Response, and statistics on the number of characters in the text responses. Scan does NOT break out the separate values in a TEX field for analysis.

SCAN also lets you do T-tests on banner columns. You can test at three different levels of specificity: .90, .95, or .99.

When specifying All Variables or All of Certain Types, it prints Number of Responses, Mean, Standard Deviation, Standard Error, Median and Number of Characters. The question type for all variables is displayed.

SCAN supports delimited and HTML output. Printable files and delimited files can be printed for import to EXCEL or data programs, or HTML output which may be read with a browser or posted on the internet. The HTML reports have

links between the Table of Contents and tables to make navigation easier.

When to use it

SCAN is an easy way to produce report-ready tables because it allows you to use pre-existing DB variables, and it includes column headings and row labels. SCAN is also useful to produce counts for variables when crossed by some other set of variables in the study.

To see specific responses to any question type on a case-by-case basis, use LIST.

How to use it

At the operating system prompt, enter:

```
SCAN
```

and follow the instructions for each screen. At a minimum, you will have to supply a data file name, answer if you have predefined variables to use, and provide a row description to scan. You can use multiple input files, as long as they are the same file type. Use commas to separate file names, or wild cards ("?" for letters, "#" for numbers, or "*" for anything) to select a group of files.

Row descriptions can be data locations, variables or a combination of both. See *2.1 HOLE* for examples of the correct syntax to use with variables. Each additional row description will create an additional table.

If you have variables defined in a database file, you can generate a report based on variables (rather than data locations). Variables are usually question names used in Survent. You can select all of one type of question (CAT, NUM, etc.), or you can use wild cards to match variable patterns (QN??, OTH*). You can also specify a group of questions by indicating the first and last question number to include.

You can press **ESC** to exit most screens, or enter `terminate` in a text field to quit SCAN.

When you have completed all the screens, SCAN displays a summary screen which indicates the options you have selected and gives you a chance to change them before you run the report.

Input files

SCAN accepts the following file types:

- CfMC System (TR)
- ASCII
- ASCII Card-image (multiple records per case)
- Binary
- 'Swapped' Binary (from other computer types)
- CfMC HP3000 SPL format (DTA)
- CfMC Phone File (FON file from Survent)

Output files

The files produced by SCAN are:

Name	Ext.	Description
datafile	SCN	print file
HTML	HTM	Web page output
Delimited	DLM	for Excel, Web Tables, browser

A SCAN table has the following headings by default:

Heading	Description
Banner:	Text describing the banner variable.
Stub:	Text describing the stub
Base:	Text describing the base variable (if applicable).
Total:	The count for the total sample. (If you have set a base or weight, the total will reflect that base or weight, not the total number of cases in the file.)
No Answer:	The number of records with no answer.
Any Response:	The number of records with any answer.

SCAN provides both banner points (column headings) and stubs (row labels).

Here is a sample of the report produced by SCAN. A Table of Contents is printed after the last table.

```
Computers for Marketing Corp. - SCAN of rrunr.tr - Page 1
ROW: QN11
BASE: Female

-COUNT-    -PERCENT-

Total                                  168       100.0%
No Answer                              117        69.6%
Any Response                            51        30.4%

Basketball hoop                         10         6.0%
Whack-a-Mole                             7         4.2%
More video games                         5         3.0%
Specific video game                      7         4.2%
Music videos                             7         4.2%
Darts                                    7         4.2%
Foosball                                 7         4.2%
More entertainment for
teenagers                                4         2.4%
More entertainment for
small children                           8         4.8%
Child care facilities                    7         4.2%
Other                                   15         8.9%
Don't know/Refused                       5         3.0%
```

Options

To change the defaults, choose one of the following options:

1 Make your own HEADER or FOOTER

Create a heading or page footer at the top and/or bottom of pages for the report.

2 Create a BANNER

A banner is made up of the headings for each of the columns in the table. You can enter your own headings, or use predefined variable names, numbers or locations,

such as REGION, QQ010.5, or [1/5.1^1-2]. SCAN always uses Total as the first column heading.

3 BASE some tables

This is a way to use only certain records from the data. You can use a data location to determine which records to include. An example would be a base of [15^1-3], which means include only those records that have a one, two or three punch in column fifteen. You can also use variables from previous Mentor runs. Enter the name of the DB file in which the variables are defined. See *2.1 HOLE* for examples of the correct syntax to use for variables.

4 WEIGHT the data

Weighting is a way to change the relative value of each record. You can give the answers from a specific group in the study more weight than answers from the rest of the respondents (for example, all the answers from males are given a value of .75). This is usually done to compare the frequencies to the numbers in the cells of weighted tables. You can also specify how many decimals of significance to print.

You can use data locations or variables to define the weight; see the examples under the BASE option of the HOLE utility for the correct syntax to use for different types of variables.

5 Specify the Print Options

To change one of the print options from the default, enter the item number or letter of the item(s) you wish to change. You must enter all the options you wish to change at once; separators between the item numbers or letters are not necessary. For example, entering 23A would allow you to add horizontal percents(2), change the number of decimals in the percents(3) and change the page length(A). A list of formatting defaults follows.

Option	Default
1) Column width, Stub width	80/20 with banner, 20/30 standard*
2) Show Horizontal Percents?	no*
3) Percent decimals/% sign	1/yes
4) Print cumulative % in banner	Yes
5) Make % Base “Any Response”?	no
6) Show frequencies only?	no
7) Show percents only?	no
8) Suppressed blank tables or rows?	no
9) Print one table per page?	no
0) Print Sigma at bottom?	no
A) Page Length/Width	66/80 (w/banner 132)
B) Suppress TOTAL/NA/Any resp.?	no (Can suppress any)
C) Title w/ variable locations?	yes
D) Include table of contents?	yes

*These are the defaults if you use Total as the only column. If you add more columns, the column width default is 8 and the stub width default is 20.

6 RANK tables high to low

This prints counts with the most occurrences first, rather than sequentially.

7 To add STATS at the bottom of the report, enter the number of the item(s) you wish to add in the order you want them displayed.

1) Mean

2) Standard Deviation

3) Standard Error

4) Sum

5) Median (50% percentile)

6) Variance

7) Minimum Value in Field

8) Maximum Value in Field

9) Stats Only (All stats across page compressed)

Note: Even though statistics only make sense on rating scales or single location numeric fields, if you select the statistics option, they will appear on all tables in your frequency report. You cannot have statistics on some tables and not on others in the same run.

8 Frequency Count tables

This produces a "frequency count" table, which is a standard format report that includes counts, number of cells per table, percents, cumulative percents, and sub-totals when doing variables within other variables. You can get up to 20 separate variables. You can to cross up to 5 variables per table.

A frequency count shows what ASCII characters are in specified locations in the data and reports the number and percentage of each group in relation to the total sample. This is especially useful to look at short, multiple-column character sets, such as numeric codes, state abbreviations, zip codes or telephone numbers. Frequency counts are not case-sensitive and will put the strings "TEST", "Test" and "test" in the same category. (If you need a frequency count that is case-sensitive, you will have to use Mentor to write a Mentor spec file and include the command ~SET CASE_SENSITIVE).

Here is a frequency count table with two variables:

```
first second  -COUNT-           -% OF TOTAL-     CUM. %
(6 cells in table)
   AA    BB         1               10.0             10.0
   AA    CC         2               20.0             30.0
   AA              --    3 30.0
   BB    BB         2               20.0             50.0
   BB    CC         2               20.0             70.0
   BB    DD         2               20.0             90.0
   BB              --    6  60.0
   EE    EE         1               10.0            100.0
   EE              --    1               10.0

                   --   10         100.0
```

For more on frequency count tables, see *Mentor, Appendix B, ~FREAK*.

9 Output options

SCAN supports delimited and HTML output. Output filesnames can be up to 14 characters, plus the extension. Printable files and delimited files can be printed for import to EXCEL or data programs, or HTML output which may be read with a browser or posted on the internet. The HTML reports have links between the Table of Contents and tables to make navigation easier. You will be presented with the following options on your screen:

```
In addition to a printable ASCII report, do you want a delimited file
(for Excel/Spreadsheets), or Web Tables (to read with a browser)? Enter
the report types you want, or "A" for "All."

1. Just a printable report

2. Delimited file (for Excel or other spreadsheets)

3. Web Tables (to read with a browser

A All types (printed, delimited and web)
```

3.3 LIST

What it does

LIST produces a listing of all responses for a particular data location or variable. The LIST report is not a summary count (if you want a summary count, use HOLE or SCAN), but a case-by-case display of the responses.

When to use it

LIST is usually used to print the responses to open-end questions. This information can be used to list all specific responses to certain questions or question types. You can also create coded categories for an item and use Mentor to code the data and create summary reports. See *Mentor Volume I, Chapter 3: REFORMATTING YOUR DATA* for information on how to set up codes and *Chapter 6: ADVANCED TABLES* for details on how to produce summary reports.

How to use it

At the operating system prompt, enter:

```
LIST
```

DOS Note: *"List" is also a Norton Utility for DOS. To ensure you are running the CfMC utility, use the full path to start LIST. In a standard installation, enter:*

```
f:\cfmc\go\list
```

Follow the instructions for each screen. At a minimum, you will have to supply a data file name, which type(s) of variables you want to list, which variables or data locations to include, and how you want the list formatted (see *OPTIONS*, below). See *2.1 HOLE* for examples of the correct syntax to use with variables. You can use multiple input files, as long as they are the same file type. Use commas to separate file names, or wild cards ("?" for letters, "#" for numbers, or "*" for anything) to select a group of files.

If you have variables defined in a database file, you can generate a report based on variables (rather than data locations). Variables are usually question names used in Survent. You can select all of one type of question (CAT, NUM, etc.), or you can use wild cards to match variable patterns (QN??, OTH*). You can also specify a

group of questions by indicating the first and last question number to include.

You can press **ESC** to exit most screens, or enter `terminate` in a text field to quit LIST.

When you have completed all the screens, LIST displays a summary screen which indicates the options you have selected and gives you a chance to change them before you run the report.

Input files

LIST accepts the following file types:

- CfMC System (TR)
- ASCII
- ASCII Card-image (multiple records per case)
- Binary
- 'Swapped' Binary (from other computer types)
- CfMC HP3000 SPL format (DTA)
- CfMC Phone File (FON file from Survent)

Output files

The files produced by LIST are:

Name	Ext.	Description
datafile	LST	print file

Sample output

A LIST report has the following items:

Item	Description
List order:	How the data is listed, either question by question or case by case.(see ***OPTIONS***, below)

Item	Description
Variable or Data Location:	The variable or data location you have chosen to list.
Description:	The question label and text.
Case:	Case Identifier (ID). For files without case IDs, LIST will default to the first ten columns of the data.
Data:	Whatever text or data appears for that variable or data location.

Instead of having as many items will fit on a page, you can choose to have each variable on a page by itself. The default page width is 80 columns.

Here is a sample report produced by LIST.

```
Computers for Marketing - Listing of responses for JEANS.TR


List order: Data case responses listed within each variable
```

```
********** [2/10] **********
Description: Q5O - OTHER JEANS USUALLY WORN
             ------------------------------

Case: 0009 , Data: 'Bossimo, Stubby'
Case: 0013 , Data: 'Badland'
Case: 0015 , Data: 'Bill Blass'
Case: 0004 , Data: 'VCW, PAVORI'
Case: 0005 , Data: 'VCW'
Case: 0009 , Data: 'PAVORI'
Case: 0007 , Data: 'LOBO JEANS'
Case: 0008 , Data: 'TOMMY SMITH, WALKERS'
Case: 0004 , Data: 'WATER POLO'
Case: 0008 , Data: 'NO NAME'
Case: 0013 , Data: 'VCW'
Case: 0001 , Data: 'PAVORI'
Case: 0004 , Data: 'DRUM/ACME/X-FILE.'
Case: 0005 , Data: 'CODE RED'

 There were 14 responses to "Q50"
```

Options

LIST also allows you to:

1 BASE the tables

This is a way to use only certain records from the data. Use a data location to determine which records to include. An example would be a base of [5.2#1-50], which means include only those records that have the numbers one through 50 in columns five and six. You can also use variables from previous Mentor runs. Enter the name of the DB file in which the variables are defined. See *2.1 HOLE* for examples of the correct syntax to use for variables.

2 Write open-end text to a file.

The open-end text can be written to a special format file to be read by a Word Processor or Spreadsheet program (e.g. WORD or EXCEL).

LIST writes files that can be read by EXCEL as follows:

```
CASE ID      LABEL     location       TEXT OF RESPONSE
```

You can manipulate the text as you see fit, spell-checking or editting for content. Then you can export the file to be read back into your CfMC TR file using the support spec file "RDTEX^SPX".

Refer also to the support spec file "DMPTX.SPX" which does the same thing using Mentor command language.

3 Print an extra variable.

Add an extra variable along with the case ID for each case, such as a name. This is useful to include another variable, such as interviewer ID, or when printing all variables of a certain type and you also want to include one variable of a different type.

4 Specify how you want the list organized:

- a variable heading and all the data cases within that variable, or
- a data case heading and all the variables from that case below it; if you choose this option, you can have the variable name, question text and response, or just have the variable name and response on the list.

5 Choose the number of blank lines between cases:

0) compressed (none)

1) single spaced

2) double spaced

6 Adjust the page width

You can have a page width from 25 to 250 characters. The default width is 80 characters.

The options menu looks similar to the other utilities:

```
Enter any LIST options you want to use or Press <Enter> to list all cases
to a print file in portrait format.

B Specify a BASE (Subgroup of cases to LIST)

O Write Open-end text to a special format file to be read by a Word
Processor or Spreadsheet program (e.g. WORD or EXCEL)

V Print an extra variable in addition to the case ID

P Start each new item on its own page

L Add blank lines between each item listed

W Change the page width/use landscape print format
```

Chapter 4
Data Alteration

4.1 CLEANIT

What it does

Cleaning is the process of changing the data in data files to make them more accurate. The CLEANIT utility allows you to display, verify and modify individual records in data files.

When to use it

CLEANIT is a utility for clients that do not have Mentor. While the CLEANIT utility is easy to use, Mentor is much more powerful and flexible. CLEANIT is a subset of the Mentor command set. If you want to write complex cleaning procedures, including conditionals (IF-THEN-ELSE) or automatically clean an entire data file at once in batch mode, you need to use Mentor.

How to use it

At the operating system prompt, enter:

```
CLEANIT
```

This starts the cleaner block, and opens a log file called CLEAN.LOG. If a file named CLEAN.LOG already exists, CLEANIT will append to this file. If you want a separate log file for each session, you must rename the existing CLEAN.LOG file.

You can also enter the single argument of a study name. Mentor will open the TR, DB and QFF files for the study name, if they exist. With the DB file open, you can display and modify questions using label names rather than data locations. With the QFF file open, you can modify text questions without having to define the text

location. For example, if your study name is bank, you can type:

```
CLEANIT bank
```

CLEANIT also supports going directly to a particular CASE ID. IF you want to start at a particular case ID in the data file (TR file), type:

```
CLEANIT <study><caseid>
```

If the case id is missing, you are placed on the second case.

You can enter `>HELP` to look at the help screens to see all the CLEANIT commands available. Enter `~END` to quit CLEANIT.

CfMC recommends the following procedure to clean data with CLEANIT:

1. **Copy the data file. (DOS/MPE "copy")**
2. **Start CLEANIT. (CLEANIT)**
3. **Open the data file, if necessary. (FILE)**
4. **Display the data. (DISPLAY ASCII, DISPLAY BINARY, DISPLAY TEXT)**
5. **Modify and check the data. (MODIFY ASCII, MODIFY BINARY, MODIFY TEXT)**
6. **Move to the next case. (NEXT)**

Input files

Data lookup or modification CLEANIT commands work on CfMC System files, ASCII and binary files. For ease of use, you can use COPYFILE or MAKECASE to convert other file types to a CfMC System file.

Sample cleaning session

1 Copy the data file

The changes you make to a data file with CLEANIT are permanent, so always have a backup of the original data. Before you start CLEANIT, use the COPY command to make a backup copy of your data file. Enter:

```
COPY file.tr back.tr (DOS, UNIX)
COPY filetr, backtr (MPE)
```

2 Start CLEANIT

Enter:

```
CLEANIT <studyname>
```

The Mentor cleaning block will display the prompt “CLeaNer command or >help-->.”

3 If you need to open a data file, enter:

```
FILE filename
```

If your file is not a CfMC System file, you must indicate the file type and length.

Enter:

```
FILE filename ASCII=<length>
FILE filename binary=<length>
```

CLEANIT automatically positions you on the first record of the file.

4 Display the data

You can display data in either ASCII and binary format. The commands are:

```
DA <location or variable>
DB <location or variable>
```

You can either use a data location or variable. Indicate a data location by entering the starting column and width. You can use two different formats, either the absolute location or record number and column number.

For example:

`90.5` (absolute column number and width)

`2/10.5` (record number/column and width)

No matter which format you use to enter data location, CLEANIT will report the data location in record number/column number and width format (2/10.5). If you do not include a record number, it will default to one. If you do not include a column width, it will default to one.

If you have a database file that contains variables, you can use variables to call data. While variables can be anything, they are usually the question names used in Survent. Before you use variables in CLEANIT, you must have a database file open. If you need to open a database file, enter:

```
>USE_DB <filename>
```

a) ASCII DATA

ASCII format is for single-punched or alphanumeric data. Since there can be only one character per column, the columns are represented vertically, that is, the data

from all columns are listed on the same line. The top two lines are a template which indicates the column numbers. For example, to look at the first forty columns of the current case, enter:

```
DA 1.40
```

CLEANIT will display:

```
ID: 0001 (study code=RRUN, int_id=bkok):1.40
Display 1/1.40:
0 1 2 3 4
12345678901234567890123456789012345678901
----------------------------------------
0001 340101011RF 3344125 1224114233
```

In this example, the case ID is in the first four columns (0001), column five is blank, column six is a three, column seven is a four, etc. To display an entire data record, you can enter "D*". To display part of a record, enter the DISPLAY_ASCII command with a data location, for example, to display just columns 10 through 16, enter:

```
DA 10.6
```

CLEANIT will display:

```
ID: 0001 (study code=RRUN, int_id=bkok):10.6
Display 1/10.6:
1
0123456
-------
01011RF
```

If there were any multipunched numeric data in this record, it would be represented with an asterisk. To be able to see multiple punches in one column, display that column in binary mode.

b) BINARY DATA

Letters, punctuation marks and multipunched numbers are stored in the data file by multiple punches in one column. Because there is a combination of characters in each column, binary data is represented horizontally; that is, each column is listed on a separate line. Each line contains the record and column location, the ASCII character code, and all of the punches in that column. To look at columns

13 through 17 of the same record as above in binary format, enter:

```
DB 13.5
```

CLEANIT will display the column number, the ASCII character and the binary data:

```
ID: 0001 (study code=RRUN, int_id=bkok):1.5
column ASCII Binary
1/13 1 1
1/14 1 1
1/15 R 9,11 (‘X’)
1/16 F 6,12 (‘Y’)
1/17
```

Here is an example from another data file that has a multipunched number in column 17:

```
DB 14.5
ID: 0002 (study code=TEST, int_id= ):14.5
column ASCII Binary
1/14 4 4
1/15 ! 2,8,11('X')
1/16
1/17 * 2,3,6
1/18 A 1,12('Y')
```

You can see from this example that letters (column 18) and punctuation marks (column 15) are also made up of multipunches.

c) TEXT DATA

Data from TEX questions from Survent are stored in compressed format at the end of the data file. Survent uses text pointers, a data location that points to another data location where the text actually is. If you don't already have your QFF file open, indicate the start of the text location when opening the data file.

The syntax is:

FILE <name> TEXTLOCATION=<location>

For example:

```
FILE mydata TEXTLOCATION=5/1
```

To display the data from a TEX question, enter the display text command (DT) and a text pointer location. For example, to display text data from the current case

that is referenced by the text pointer in record 4, column 71, enter:

```
DT 4/71
```

To list all the text from the current case, enter:

```
DT *
```

If you want to list the responses for a particular data location from every case in the data file, you can use the LIST utility.

5 Modify and check the data

a) (ASCII)

Data can be modified in either ASCII or binary mode. Use ASCII mode unless the data is multipunched. Enter the modify ASCII command (MA) and a data location. Enter:

```
MA 2
0.5
```

And CLEANIT will display:

```
ID: 0001 (study code=RRUN, int_id=bkok):20.5
Display 1/20.5:
2
01234
-----
33441
MA->
```

To change the data, enter a character for each column. If you want to make a column blank, then enter a space for that column. If no changes are necessary, or no changes beyond a certain point, press Enter. For example, if you want to keep the first column a number three, change the second column to a blank and change the third column to the number one, at the “MA->“ prompt, enter:

```
3 1
```

Now, check your changes by displaying the data in the same location again. Enter:

```
DA 20.5
```

And CLEANIT will display:

```
ID: 0001 (study code=RRUN, int_id=bkok):2.5
Display 1/20.5:
2
01234
-----
3 141
```

NOTE: *Once you reach the modify ASCII (MA) prompt, you must enter something for each column, or press the Enter key to leave modify mode. Entering a blank with a space bar changes the data to a space, it does not keep the data the same.*

You cannot add data past the last column in the data record. Even if the data record is defined as 80 columns long, but you only have data up to column 40, you cannot add data to columns 41-80.

b) BINARY:

Letters, punctuation marks, and non-ASCII numbers are all stored in the data bya combination of punches in a single column. When displaying data in ASCII mode, asterisks represent numeric columns with more than one punch. To see all of the punches in the column, you must use the display binary or the modify binary commands. To display the data in columns 14 through 18 in the current case, Enter:

```
DB 14.5
```

CLEANIT will display:

```
ID: 0002 (study code=TEST, int_id= ):14.5
column ASCII Binary
1/14 4 4
1/15 ! 2,8,11('X')
1/16
1/17 * 2,3,6
1/18 A 1,12('Y')
```

To change the data, use the modify binary command (MB) and the data location, Enter:

```
MB 14.5
```

CLEANIT will display the first column and prompt you for new data:

```
ID: 0002 (study code=TEST, int_id= ):14.5
column ASCII Binary ....New Data
1/14 4 4 MB->
```

You must enter something at each "MB->" prompt, either new data or press Enter to keep it the same. You can enter a single punch (1234567890XY) or multiple punches separated by commas (e.g. 7,8,9). To keep the current punches and add a punch, enter a plus sign and the punch you wish to add (e.g. +2), or to remove a punch enter a negative sign and the punch you wish to remove (e.g. -2). To make a column blank, enter B. So, to leave column 14 the same, change column 15 to blank, change column 16 to punches one and five, add a five punch to column 17 and leave column 18 the same, you would Enter:

```
ENTER; B; 1,5; +5; ENTER.
```

It would look like this:

```
ID:0002(study code=TEST, int_id= ):14.5
column ASCII Binary ....New Data
1/14 4 4 MB->
1/15 ! 2,8,11('X') MB->B
1/16 MB->1,5
1/17 * 2,3,6 MB->+5
1/18 A 1,12('Y') MB->
```

Now, review your changes by displaying the data in the same location again. Enter:

```
DB 14.5
```

And CLEANIT will display:

```
ID:0002(study code=TEST, int_id= ):14.5
column ASCII Binary
1/14 4 4
1/15
1/16 * 1,5
1/17 * 2,3,5,6
1/18 A 1,12('Y')
```

You could use ASCII mode to change multipunch columns of data to single punch, but you would not be able to see exactly what data was in the multi-punched columns before you change it.

c) TEXT

To modify the data from a TEX question, enter the modify text command (MT) and

a text pointer location. For example, to display text data from the current case that is referenced by the text pointer in record 4, column 71, enter:

```
MT 4/71
```

CLEANIT will display:

```
ID: 0016 (study code=RRUN, int_id=bkok):[4/71] =Roadrunner's
pizza is totally rad, dude!
```

Then, CLEANIT will display a second screen which has the text in a box. You can now use the text editor to modify the text. Press ESC when you are finished. Display the text again to check your work, enter:

```
DT 4/71
```

CLEANIT will display:

```
ID: 0016 (study code=RRUN, int_id=bkok):[4/71] =Roadrunner's
pizza is totally rad, dude!
```

6 Move to the next case

When you use FILE to open a data file, you are positioned on the first case in the file. To move to the next case, enter:

NEXT

You can move forward by entering NEXT and a plus sign and the number of cases forward you want to go. For example, to remove four cases, Enter:

```
NEXT +4
```

NOTE: *Cases are stored in the data file in the order they are entered; cases do not necessarily have to be in case ID order. Do not assume that because you are the fourth case in the data file and you move forward four cases that you will be on case ID 8.*

To move to a specific case, enter `NEXT` and the case ID. You must include leading zeros, for example, if your case ID is four columns wide and you want case ID

five, enter:

```
NEXT 0005 or N"0005"
```

You can also enter `NEXT FIRST` and `NEXT LAST` to go to the first and last cases, respectively. To move backwards in the data file, use the BACK command. To move three cases back, Enter:

```
BACK 3
```

Other cleaning commands

Enter `>HELP` to get a listing and short description of cleaning commands.

Deleting a case

To delete a case, DELETE. UNDELETE will remove the delete flag from the case.

Repeating commands

CLEANIT allows you to repeat simple procedures easily. You can put several commands on one line separated by semicolons. The NEXT_REDO command will re-execute the entire line. For example, if you want to modify the data in columns one through four and display the same columns again (to check your changes) and then do the same thing on the next case, you would Enter:

```
MA 1.4; DA 1.4
NR
```

Using semicolons between commands can lead to long command lines. To make changes to the line, use the meta command >EDIT_PREVIOUS. Enter:

```
>EP
```

When you are through with your corrections, press ESC and CLEANIT will execute the new command(s).

Finding a case

To find data cases that meet a criterion, use the HUNT or FIND commands. HUNT starts at the beginning of the data file and stops at the first case that satisfies the criterion, while FIND starts from your current position and searches forward in the file. For example, to find the next case that has a one punch in column

five, enter:

```
FIND [5^1]
```

To find the next case that matches the same criteria, just enter `FIND` or F.

Showing variables

If you have variables in a database file, you can use the SHOW command to the complete variable (the question text) and the answer. To use variables, you must have a database file open. Enter:

```
>USE_DB <filename>
```

Then use SHOW with name of the variable you want to see. For example:

`SHOW qn18` and CLEANIT will display:

```
qn18( 402,1 )($C: 7) title=Q18. Would you classify your ethnic
background as . . . [ 1/53 CAT ]
categories: #1: (1) Caucasian #2: (2) African-American #3: (3)
Hispanic #4: (4) Oriental #5: (5) American Indian #6: (9) Some
other ethnic group #7: (0)Refused=qn18(7:4=4=Oriental)
```

Defining procedures

You can use HUNT and FIND (described above) with defined procedures. You can write a short procedure and use it on the entire data file. For example, to modify and review the ASCII data in columns five through seven, you could define the following procedure once you are in the cleaning block:

```
DEFINE PROCEDURE=REPAIR:
MA 5.2
DA 5.2
}
```

Once the procedure has been defined you can execute it any time during the current cleaning session by entering:

```
!REPAIR
```

Or you could start at the beginning of the data file and execute REPAIR on every case.

Enter:

```
HUNT REPAIR
```

Restoring a case

If you make changes to a data case and do not want to save the changes, enter:

```
RESTORE
```

This will restore the case from the last time it was saved to disk. When you move to another case, the case is stored to disk and you cannot reverse those changes.

Viewing a questionnaire

Use the SURVIEW utility to view a data file in relation to a questionnaire file. You must have a Survent questionnaire file (this file has a qff extension) and a corresponding data file. Open the data file, the questionnaire file and then start SURVIEW:

```
FILE datafile
~QFFFILE questionnaire file
~CLEAN VIEW
```

See *Chapter 4: CONDUCTING THE INTERVIEW* of the Survent manual for details on how to use SURVIEW.

Modifying case IDs

The case ID is a record identification number from one to ten characters long. Each data record has both an assigned and internal case ID. The assigned case ID is created by Survent (or another program) and is contained in data location assigned by the user.

Often assigned case IDs are four digit numbers kept in columns one through four. The internal case ID is a number assigned automatically by Survent or Mentor and resides in the case header, which is not part of the regular data.

To change a case ID, you must change *both* the assigned case ID and the internal case ID. To change the assigned case, use the MODIFY_ASCII command. To change the internal case ID, use the PUT_ID command. For example, here are the steps to change case ID 0005 to case ID 0999.

First, display the case. Enter:

```
DA 1.20
```

CLEANIT will display the internal case ID on the first line:

```
ID: 0005 (study code=RRUN, int_id=bkok):1.640
Display 1/1.20
0 1 2
12345678901234567890
--------------------
0005 033530205 5 3
```

To change the assigned case ID, change the data in the location (in this example, columns one through four) with the MODIFY_ASCII command. Enter:

```
MA 1.4
```

CLEANIT will prompt you for the new data:

```
ID: 0005 (study code=RRUN, int_id=bkok):1.4
Display 1/1.4:
0
1234
----
0005
```

Enter the new data:

```
0999
```

Now, move the new assigned ID from the assigned data location into the case header with the PUT_ID command. This command requires a string type variable, so you must put a dollar sign ($) after the data location. Enter:

```
PUT_ID [1.4$]
```

Display the case again to verify the new case IDs in both the case header and the data location. Enter:

```
DA 1.20
```

And CLEANIT will display:

```
ID: 0999 (study code=RRUN, int_id=bkok):1.640
Display 1/1.20
0 1 2
12345678901234567890
--------------------
0999 033530205 5 3
```

Or, you can use the SAY CASE_ID command to see the current internal case ID.

Enter:

```
SAY CASE_ID
```

And CLEANIT will display:

```
0999
```

If you want to write more complex cleaning procedures, including conditionals (IF-THEN-ELSE) or automatically clean an entire data file at once in batch mode, you need to look at all the features you can get with Mentor.

These commands are described in detail in *Mentor Volume I: Chapter 2: PREPARING YOUR DATA* and *Mentor Appendix B: ~CLEA*N.

4.2 CODEEDIT

What it does

The CODEEDIT utility allows users who are willing to follow particular questionnaire-writing rules to have a complete coding solution. It is the best solution for users who have a coding department and a coding manager who can edit code lists to keep multiple coders working using Survent under the server.

CODEEDIT is supported on all platforms. It reads a QSP file and generates a coding and editing questionnaire.

When to use it

You should label questions to be coded with label extensions such as OE or OTH (questions with OTH extension will have code lists associated with them).

The program runs in two passes: First, it finds the Other Specifies and open ends and makes a file out of each. The user then adds codes to these files or changes their location, etc. Then the program is run again and the updated code lists are used to make a coding questionnaire.

The coding questionnaire has the following options:

- Code an uncoded question across cases.
- Code all questions on uncoded cases only (if the case is previously untouched).
- Edit particular open-end questions or all questions.
- Or, get a particular case to code whether or not it previously has been coded.

Other options

- **codeedit moveothercodes:** This moves codes from the original controlling questions for “other-specifies” to the same location as the coded data so that it is all in the same location. This is done at the end of coding.
- **“codeedit counts”** This displays how many are coded/not coded for each question.
- **“listcode <label>”:** This lists a;; the cases tha need coding for a partiucular question and the coded values if coded in tab-delimited format. It also gives a summary count for the question.

How to use it

Type:

```
codeedit
```

You will get this main menu:

```
WELCOME TO THE AUTOMATIC CODING AND EDITING PROGRAM!
WHAT DO YOU WANT TO DO?
```

Type F2 then "TERM " at any time to drop changes and return to the main menu.

```
1 Code one "other specify" question (UNCODED CASES ONLY)
2 Code one "open-end" question (UNCODED CASES ONLY)
3 Edit a particular "other specify" across cases
4 Edit a particular "open-end" across cases
5 Code ALL others and openends within each case
6 Edit ALL others and openends within each case
X ** EXIT PROGRAM **
```

4.3 VERBEDIT

What it is

VERBEDIT is similar to the CODEEDIT utility, except it only allows editing of verbatims.

How to use it

You need only specify the extension of each question to be included and run the

program to create a questionnaire file to be used for editing. You can start at a particular case or just get the next case to be edited. The time and interviewer ID of the editor is stored in the dataset.

Here is the main menu:

```
WELCOME TO THE AUTOMATIC VERBATIM EDITING PROGRAM!
WHAT DO YOU WANT TO DO?
```

(NOTE: Type F2 then “TERM” to drop changes and return to main menu.)

```
1) Edit a particular open-end question across cases
2) Edit ALL verbatims within each case
```

Chapter 5
Customizing CfMC Software

5.1 CFMCMENU

What it does

CFMCMENU is a menu-driven front end to CfMC software.

When to use it

This is for anyone who prefers a menu-driven front-end to CfMC software. Advanced users can use the programmer's menu, see *OPTIONS* below.

How to use it

You can work on projects that are not currently in the field or you can access studies that are running live under the CfMC SERVER. Enter:

```
CfMCMENU
```

Note: Menu options will vary according to the platform on which you are working. The following menus reflect a DOS platform.

If you choose the first option, to work on studies that are not currently live, the following menu will appear:

```
                    ***** CfMC Software Menu System *****

Survent / System Utilities                Data Reporting Utilities
---------------------------   ----------------------------------
1) Run Survent/test questionnaire      5) SCAN (tables for category data)
2) Quotamod (report/update quota info) 6) FREQ (tables for number/var data)
3) Suspres  (report suspend info)      7) Holecount (summary column count)
4) Foneutil (report/update phone info) 8) LIST (show answers case-by-case)

Phone System           Data Review/Alteration        New File Creation
--------------       ----------------------    ---------------------
F) Make new sample   I) Cleanit (see/update data)  L) Copyfile (many options)
G) Check phone file  J) View interview (single)    M) Reformat (spread file)
H) Do Phone Reports  K) Recode data (single user)  N) Merge (combine files)

        DOS Commands                          General
   ------------------------             -------------------------
   S) Change directories                W) Run a Mentor batch job
   T) Edit a file                       X) Exit menu, go to DOS
   U) Get listing of files              Y) Get HELP on 1-Z
   V) Do a DOS copy command             Z) Reset CfMC variables
```

If you choose the second option, to work on studies while they are live, the same menu will appear, except for the first section. Instead of "Survent/System Utilities," you will see:

```
Live data collection
  ------------------------------
  1) Supervise/start interviewers
  2) Monitor (interviewers)
  3) View (review/update data)
  4) Interview (under server)
```

Options

There is an advanced version of CFMCMENU for programmers called PROGMENU.

Enter:

```
PROGMENU
```

and you will see a menu that resembles the following:

```
                                                                  ***** CfMC
Software Menu System *****
Questionnaire
Development           Live Data Collection        Live Survent Sub-systems
----------------    --------------------------  --------------
1) Edit spec file   6) Supervise/start intvwrs   A) Quotamod (quota updt/info)
2) Compile qnaire   7) Monitor (interviewers)    B) Suspres  (suspend info)
3) Test/run qnaire  8) View (review/update data) C) Fixresum (fix suspends)
4) Convert document 9) Interview (under server)  D) Foneutil (phone updt/info)
5) EZWriter         0) Cleanit (updt live data)  E) Makecfg  (stations info)

Phone System         Data Review/Alteration         New File Creation
-----------------   --------------------------  -------------
F) Make new sample  I) Cleanit (see/update data) L) Copyfile (many options)
G) Check phone file J) View interview (single)   M) Reformat (spread file)
H) Phone reports    K) Recode data (single user) N) Merge (combine files)

Data Reporting          DOS Commands                  General
------------------  ------------------------     ----------------------
O) Scan (tables)    S) Change directory          W) Run a Mentor batch job
P) Freq (num tabs)  T) Edit a file               X) Exit menu, go to DOS
Q) Hole count       U) Directory of files        Y) Get HELP on 1-Z
R) List open-ends   V) Do a DOS copy command     Z) Reset CfMC variables
```

Both CFMCMENU and PROGMENU are batch files in the \cfmc\go directory

which you can customize for your own needs.

The Survent utilities QUOTAMOD, SUSPRES, FIXRESUM, FONEUTIL, MAKECFG, RECODE, and REFORMAT are documented in the *Survent* manual. The utilities COPYFILE, MERGE, SCAN, FREQ, HOLE, LIST and CLEANIT each have their own section in this manual.

5.2 MAKEMSG

What it does

MAKEMSG allows you to customize the program messages and help screens that appear while running Survent, Mentor and Script Composer.

This utility supports message file lines that contain up to 300 characters. If you add long messages, be careful because you must need to pay attention to line wrapping on a terminal at 80 columns when messages are displayed.

When to use it

Use MAKEMSG to customize informational or error messages. You can also use MAKEMSG to have your messages appear in another language. Currently Spanish, French and German are supported.

How to use it

All program messages are contained in the message file. To change a message, make a copy of the ASCII version of the message file (\cfmc\control\msgfile.raw), and then edit it to include the changes you want. Compile the file by entering:

```
makemsg
```

You will be prompted for the name of the file you have edited. MAKEMSG will compile create a compiled message file with the name "newfile." Rename newfile to msgfile and copy it to \cfmc\control directory. Survent, Mentor and Script Composer will now use the messages you have added to the message file.

Which message file?

CfMC programs will first look for a message file named msgXXXX, the Xs being a four-digit suffix that is the current program version number (for example, msg9607). If the program does not find a msgXXXX file, then it will use the file

named "msgfile." To see which message file your CfMC program is currently using, use the meta command >VERSION.

To change your messages to another language, see the OPTIONS section below.

Options

Changing the language

If you want to change your messages to Spanish, French, or German, follow these steps (some basic messages have already been translated for you):

1 Change the message

Place the message in the message file (msgfile.raw) just before the English version. Start the line with two letters indicating the language (sp=Spanish, fr=French, ge=German), keeping the message number the same. Do not change any variables, such as %s or %d, as they are filled in by the program. For example:

```
g4377zu viele Antworte (nur %d anerkannt)
4377too many responses (only %d allowed)
```

2 Make a new message file

Enter:

```
MAKEMSG msgfile.raw gapfile language=ge
```

Msgfile.raw is the ASCII version of the message file that you have edited in step one. While MAKEMSG creates the new message file, it will keep a record of any gaps in the numbering sequence of the program messages in the file name you indicate with "gapfile." Commonly used language options are sp=Spanish, fr=French, and ge=German, respectively. They are represented by two characters.

You can set your own language abbreviations in your header. But, it is best to abbreviate languages so that they are recognizable.

CfMC supports 350 languages. Here are some abbreviations for some of the commonly used languages:

- GE: German
- EN: English

- FR: French
- IT: Italian
- PT: Portuguese
- SP: Spanish
- SV: Swedish
- ZH: Chinese
- XX: All languages

Using this option will compile msgfile.raw into a file called newfile. Rename newfile to msgfile.raw and make sure it is in the \cfmc\control directory. Survent, Mentor and Script Composer will now present the messages in the language you have added to the message file.

The MAKEMSG file allows lines up to 300 characters wide.

Make a small MSGFILE for Survent on a diskette

The MAKEMSG option SMALSURV only puts message in the message file relevant to standalone Survent. This is for disks by mail.

The full syntax for the MAKEMSG command line is:

```
MAKEMSG <rawfile><listfile>SMALSURV LANGUAGE=<letters>
```

Input files

You cannot edit the msgfile directly. Make a copy of \cfmc\control\msgfile.raw and edit it. You can edit the copy with any ASCII editor.

Output files

Newfile is the compiled version of the message file (it needs to be renamed msgfile). Gapfile is a record of any gaps in the sequence of message numbers MAKEMSG encounters when compiling the message file.

Points to note

Error messages: When MAKEMSG gives informational messages while compiling the message file and finishes with the error:

```
ERROR (2602), but no message file available!
```

This is normal and does *not* indicate a problem.

MAKEMSG requires TERMTYPE=9 on a MPEXL platform: MAKEMSG makes new msgfiles for CfMC programs. In MPEXL, to run MAKEMSG and create a msgfile you must first type:

```
setvar TERMTYPE 9
```

This was done to keep unauthorized users from making msgfiles.

Appendix A
Meta Commands

RULES OF META COMMANDS

Meta commands are high-level commands, many of which work in both Survent and Mentor. Meta commands are preceded by a greater than (>) character. When the program reads this > sign, it knows that a special program command is being requested. Appendix C of the Survent Manual lists the meta commands specific to that package.
Meta commands that work in both Survent and Mentor are presented here in alphabetical order.

Here are the general rules for meta commands:

- Meta commands must start in column one. (Unless you use >ALLOW_INDENT.)
- Meta commands are used to turn options on and off by placing a minus sign (-) after the greater than sign (>) and before the command.

  For example: >-ALLOW_INDENT.
- Options to meta commands can be entered in any order (unless otherwise indicated).
- The command as shown on the first line of each section (in the left margin) reflects the full version of the command. See the syntax for the correct abbreviation.
- Underscores used to separate words are always optional. For example: DB_TO_FILE can be abbreviated DBTOFILE.
- Plural commands can be abbreviated without the S. For example, COLORS can be shortened to COLOR.

- Standard abbreviations are:

CLEAN	CLN
COLUMN	COL
ERROR	ERR
LENGTH	LEN
TOTAL	TOT
WIDTH	WID
WEIGHT	WT

See *Appendix B: ALLOWED ABBREVIATIONS* for the specific abbreviation for all CfMC program commands.

META COMMANDS AND THEIR SYNTAXES

>ALLOW_INDENT

Allows meta commands, comments and &filename commands to be indented; by default, they must start at the first column of the file. You can turn off this command by specifying >ALLOW_INDENT.

Syntax:

```
>ALLOWIND
```

>BATCH_JOB

Tells DOS- or UNIX-based systems to run the job in batch mode and not to expect input from the keyboard. For example, messages that say “Press any key to continue” would not stop batch mode processing.

>BEEP

Sets up to three beeps, specified with three sets of two numbers (#) each. The first # in each set is the tone (0 is silence). The second # in each set is the duration in milliseconds.

Syntax:

```
>BEEP # # # # # #
```

To hear the beep you've set, enter >BEEP.

>BROWSE

Displays a file on the screen, pausing when a screen is full.

Syntax:

```
>BROWSE filename
```

You can page up and down, go to the top or bottom or stop browsing at any time.

>CALL_DOS

Allows DOS commands to be called from within a program. Return to the program by typing EXIT at the DOS prompt.

Syntax:

```
>DOS
```

You can execute DOS commands and small programs, but CfMC programs remain resident in memory. You can also do a DOS command but remain in the program. See >SYSTEM.

>CASE_SENSITIVE_<filename>

This command allows you to run jobs in directories with names such as /home/study/TEST/One. It is to be used with caution if there are very long file names.

Syntax:

```
>CASE_SENSITIVE_<filename>
```

When you use CASE_SENSITIVE, the filenames, whether upper- or lower-case, will remain the same. So, your specs can be:

```
~In store/producesection/apples
or
~IN STORE/PRODUCESECTION/APPLES
or
~In Store/Producesection/Apples
```

This command keeps the case the way it was entered. It will not change the file-

name to the default, which is all lowercase. Thus, each of the three statements would look for the exact filename.

In addition, the software can read all filenames and paths regardless of the length or case (Use " "s around filenames with special characters or spaces). The program will lowercase all input and output names you specify unless you use ">CASESENSITIVE", in which case the program, will look for and write names exactly as specified.

In ALL cases, lowercase file extensionsare output, unless otherwise specified. It is assumed that setenv ENVIRONMENT variables are uppercase unless you specify ">CASESENSITIVE_ENV", in which case the program pays attention to the exact case.

For example, when turned on, files referenced, such as "Study.db", will look for "Study.db" and not "study.db". By default, it will look for "study.db" or "Study.db" or "STUDY.DB".

>CFMC_FILE_EXTENSION

This says that CfMC programs expect certain program-generated file extensions, such as TR for input files. This is the default.
It allows the user to definte alternate extensions for those usually used with files created by CfMC.
You can specify $filename on any program command (e.g., ~INPUT $myfile) to tell a CfMC program to accept the filename as given and not to add or check for standard CfMC extensions. This applies only for files with an expected default extension: data (TR); print (PRT); or db (DB). Where file type can be checked (currently only on MPE) the program will check and complain if there is a conflict. This allows you to specify your own file names up to the maximum allowed by your operating system (eight plus three character extension for DOS, eight for MPE, and 14 for UNIX (some UNIX versions have no character limit).
>CFMC_FILE_EXTENSION makes $filename the default.

The current list of 16 controllable extensions are: CHK, DB, DEF, DLM, HRD, LAB, LPR, PRT, QFF, QSP, QUO, RFL, RFT, SUM, TAB, TR

>CHARACTER_SET=ASCII (=EXTENDED_ASCII, =MULTIBYTE, =SHIFT_JIS)

This is used to say what character set is allowed in spec files.

Multibyte is used for Chinese and Korean.
Shift_jis is used to allow Japanese.

Syntaxes:

```
>CHARACTER_SET=MULTIBYTE
>CHARACTER_SET=SHIFT_JIS
>CHARACTER_SET=ASCII
>CHARACTER_SET=EXTENDED_ASCII
```

Extended_ASCII allows the use of the extended ASCII character set in spec files. This is useful for printing graphic characters, such as lines (ASCII 196) or the British pound sign (ASCII 156) in tables.
By default, only ASCII characters 32 to 126 are passed on to list and print files, while the other ASCII characters (0-31 and 127-255) are converted to spaces as the spec file is read. When this meta command is set, characters having an ASCII value of 32 through 255 are passed to the list and print files.
Regardless of the setting of the >CHARACTER_SET=EXTENDED_ASCII, tabs and characters with an ASCII value less than 32 will always be converted to a single space in list and print files. Tabs are converted to spaces so you can use tabs to organize spec files better visually, but will not be passed to the list and print files.

Consult your operating system documentation for listing of the ASCII character set.
See also >TAB_WARN.

>CHECK_EXIST

Indicates whether a file exists or not. You can give a file name in the current directory or an absolute or relative path name.

Syntax:

```
>CHECK_EXIST <filename>
```

CHECK_EXIST will generate an error and exit if a file exists, but the file name is not acceptable to Mentor, such as file names with special characters (!, -, &) in them. For example in DOS, CHECK_EXIST would find a file named junk, but will exit with a page fault with files named j--k or j!!k.

>CLEAR_SCREEN

Clears your screen. This is similar to a DOS CLS command, but will not affect

any color settings you might have in effect.

Syntax:

```
>CLS
```

>CLOSE_DB

Closes the named DB file. If all "save to" DB files are closed, items will be saved to the program's local DB file.

Syntax:

```
>CLOSEDB filename
```

This only closes the last DB file opened with >CREATE_DB or USE_DB. See also >RESET_DB.

>CLOSE_QFF

Closes open QFF files after a compile, or after a file has been opened with the ~QFF_FILE or &&&qfffile command.

>COLORS

Controls the making of color specifications when using Script Composer, and the system color when working in any CfMC program.

Syntax:

```
>COLOR CfbCfbCfb
```

Cfb Controls the system default colors on a color monitor. The color specification starts with a 'C', then has the character color letter, then the background color letter. You can specify three different color specifications on this command: the screen colors, the color for question text, and the color for response text. The foreground and background colors can be any of the following colors:

Z	black	B	blue
W	white	Y	brown
R	red	C	cyan
G	green	M	magenta

Example:

```
>COLOR CRG
```

This will print red text on a green background.

Example:

```
>COLOR CRGCBW
```

You can use a plus sign(+) before the color specifications for a bold foreground.

Example:

```
>COLOR C+WB
```

This will print bold white on blue. Using the plus sign(+) with “Y” (brown) turns it to a bold yellow.

>COLORS sets the default colors for Survent interviewing or CfMC Utility programs. This command can be used in the INITIAL file (\CFMC\CONTROL or CONTROL.CFMC) or at a CfMC program command line. This command can also be used to specify a default for all questions composed in Script Composer. The question text screen would always be displayed in the colors specified as the second and third color specs.
You can also set colors by specifying a COLOR environment variable (SET COLORS=CWGCWG in DOS), but >COLORS overrides SET COLORS. Do not specify the same color for foreground and background, because this will make the text indistinguishable from the background and you will not be able see commands as you enter them.

>CONTROL_Y_QUIET

This is for Mentor only.
It determines if you get a “DO YOU WANT TO TERMINATE?” prompt when you enter a Ctrl-Y during a Mentor run or not. By default, you are prompted to confirm termination of a run when you enter Ctrl-Y.

Syntax:

```
>CTRLY
```

Example:

```
Case # 35, ID: 0035
Mentor 12May93 running tabs.spx at 17 MAY 1993 16:24.
total time: 0 minutes 12 seconds
DO YOU WANT TO TERMINATE (No/Yes)? -->
```

Enter N, Y or ENTER (ENTER is the same as N).

With >CONTROL_Y_QUIET set, Ctrl-Y is ignored. A possible consequence is that you may not be able to terminate a Mentor run unless you reboot your PC or, on the HP, you have the system manager abort your session. On PCs, Ctrl-C may work to terminate your run.

>COPY

Copies a file. This overrides any operating system level COPY command. If you want to use your operating system version of the copy, use >SYSTEM copy. (See the >SYSTEM command.)

Syntax:

```
>COPY oldfile,newfile
```

>CREATE_DB

Creates a DB file to store question entries, variables and table specifications. After the DB file is created, it is kept open with read-write access.

Syntax:

```
>CREATEDB newdbname,options,D=dupoption,S=sizoption
```

Options:

ENTRIES= Maximum number of entries allowed. The default is 505. (E)

ECHO Displays a message when you get an item from the DB file; default is to not echo. (EC)

TEST A test mode file is one with a directory only, it is zero bytes large.

MAYBE_CREATE If the DB file exists in the current directory, a new one will not be created, and instead the existing one will be opened in ReadWrite Mode. (MCR)

DUPLICATE=dupoption Determines what the program will do when it encounters duplicate names. (D)

dupoptions:

ERROR Prevents you from storing a duplicate item and displays an error message (this is the default).

QUIET Replaces the existing entry with the new one of the same name, without displaying a warning.

REPLACE Replaces the existing entry with a warning (like Duplicate=Warn) but helps keep your db file from getting filled up with multiple versions of items since it deletes the old entry and, if the new entry is the same size or smaller, uses that same space again.

WARN Replaces the existing entry with the new one of the same name, but displays a warning message.

SIZES=sizoption Lets you specify the number of entries (ENTRIES=) by using some preset values rather thanhaving to fill in the values. The sizoptions are: SMALL, MEDIUM, LARGE, LOCAL and PREPARE. The values for each of these is SMALL=500, MEDIUM=1000, LARGE=5000, LOCAL=1000, PREPARE=3000. To change the settings for these sizes, use the >DB_SIZES command.

Example:

```
>CREATEDB newdb, EC, D=WARN, S=LARGE
```

This creates a DB file called newdb with the LARGE number of entries (5000), will generate a message when you get an item from the DB file, and will display a warning when replacing existing items.

If the DB files you create are usually the same size, put the >DB_SIZES command in your init or mentinit file instead of having to put it in your spec file every time.

You can have multiple DB files open with write access and can put items directly into any of the ‘save to’ DB files. You can have a maximum of ten DB files open at any one time. Use >RESET_DB to close all DB files and >CLOSE_DB to close one DB file.

Example:

```
>CREATEDB db1,D=W
>CREATEDB db2,D=W
>CREATEDB db3,D=W
~DEFINE-
 db1^loc1:[1/5^1.5]
 db2^loc2:[1/6^1.5]
 db3^loc3:[1/7^1.5]
>LISTDB db1
LOC1 `' type = 1, g1=0026, g2= 15, g3=  0, ver = 1
>LISTDB db2
LOC2 `' type = 1, g1=0026, g2= 15, g3=  0, ver = 1
>LISTDB db3
LOC3 `' type = 1, g1=0026, g2= 15, g3=  0, ver = 1
```

"db1^loc1" directs the variable loc1 to the DB file db1.
See also: >USE_DB, >DB_SIZES, >CLOSE_DB, and >RESET_DB.

>DB_SIZES=

Sets default values for the number of entries in DB files when you use the >CREATE_DB SIZES=option.

Syntax: >DBSIZE= e1, e2, e3, e4, e5

You must enter five numbers, and follow each number with a zero as a place holder. Each number corresponds to the SMALL, MEDIUM, LARGE, LOCAL, and PREPARE options to the SIZES= option of the >CREATE_DB command. The default values for the number of entries are SMALL=500, MEDIUM=1000, LARGE=5000, LOCAL=1000, PREPARE=3000. Use zero if you want to keep a number at the default.

Example:

```
>DBSIZE= 0 0, 0 0, 10000 0, 0 0, 0 0
```

This example will change the number of entries for a LARGE DB file from 5000 to 10000, and leaves the rest of the numbers the same.
Since the local DB file is not opened until you attempt to make an item when no other DB files are open (in read/write mode), you can control the size and duplicate parameters of the local DB file before it is opened.

If you are not creating your own DB files, you can set these numbers so that they create large LOCAL (default system) DB files which are less likely to run out of room. Keep in mind, however, that larger DB files will slow down Mentor's setup time. The LOCAL default also affects the number of tables CfMC utilities can create in one run. To allow the utilities to create approximately 5000 tables per run, set the LOCAL value to 50,000, enter:

Example:

```
>DBSIZE= 0 0, 0 0, 0 0, 50000 0, 0 0
```

For DB size LOCAL you can set any of the DUPLICATE= options by specifying the letter of the option after the second number.

Example:

```
>DBSIZE= 0 0, 0 0, 0 0, 2000 0 w, 0 0
```

The previous example opens the local DB file with DUPLICATE=WARN, overriding the default DUPLICATE=ERROR. The local DB is also increased to 2000 entries; all DB files are opened with the default number of entries listed above. Specify only the letter of the option and not DUPLICATE=.
Options are: Q for quiet, R for replace, W for warn, or E for error (default). *See >CREATE_DB for more information on these options.*
For DB files other than LOCAL, you must set the duplicate option with either the >CREATE_DB or >USE_DB command.
A default >CREATE_DB with no specifications for ENTRIES= or SIZES= makes a DB size of SMALL. ~PREPARE COMPILE creates a DB size of PREPARE.
You can also include >DB_SIZES in your MENTINIT file. This will cause the DB settings you specify to be in effect any time you run Mentor or Survent.
Use >-DB_SIZES to set number of entries back to the default values.

>DB_STATUS

Displays a list of currently open DB files in search order, noting which are ReadOnly and which are ReadWrite. DB files opened ReadWrite indicate ReadWrite access while ReadOnly files are listed by name only.

Syntax:

```
>DBSTATUS
```

Example:

```
>DBSTATUS
There are three of a maximum of ten DB files opened. Listed below are the
DB files open. They are listed in search order.
c:\acme\j1357\db3.db with ReadWrite access
'LOCAL' DB file with ReadWrite access
c:\acme\j1357\db2.db
c:\acme\j1357\db1.db
```

Now, in Version 8.1, >dbstatus reports the number of used and available entries in all open dbfiles.

Example:

```
>dbstatus
There are 2 of a maximum of 10 DB files opened
Listed below are the DB files open
They are listed in search order.

/fixed/mt3941b.db with Read-Write access (total entries=10000, used = 20)
/fixed/mt3941a.db with Read-Write access (total entries=10000, used = 20)
```

>DB_TO_FILE

Writes the DB entry to an ASCII format file. You must have a DB file open to use this command (see >USE_DB). This works for ASCII type entries only (i.e., titles, labels), DB item type=123.

Syntax:

```
>DBTOFILE  filename dbentryname
```

Example:

```
>DBTOFILE ban1 banner1_bn
```

In this example, the DB entry BANNER1_bn is sent to the text file BAN1. See also >FILE_TO_DB for the reverse procedure.

>DEFINE

Creates keywords which can be used to instruct a program to read a file, a DB entry or command(s). Enter the variable from the command line of a program preceded by the at sign (@).

Syntax:

```
>DEF @keyword foption
```

Options:

@keywordis the keyword.

foptions: can be one of the following:

```
command
&specfile
&&dbentry
```

Once a keyword is assigned to some operation, it can be used at any time during a program session to execute the operation. The keyword is not case-sensitive.

Example:

```
>DEF @HEADER &header
```

Where &header is a file containing PREPARE header statement options:
ERROR_BEEP,ALLOW_ABORT,SPEC_WIDTH=132,&

Example:

```
[STUDY1,1000,"study comment",&
@HEADER
]
```

You could create a file of definitions for specifications you use often, and either read this file in at the top of your specifications file or put it in the either the INITIAL or MENTINIT files which are read each time you access a CfMC program. These files are located in the CfMC CONTROL directory or group.

Example:

```
>DEF @STUDY sample
```

Define the study name at the top of your specifications file, then reference @STUDY in place of the study name on any meta or tilde command throughout your file.
You can append letters and/or numbers to @STUDY on a particular command, e.g., >PRINT_FILE @STUDY~2 to open the print file SAMPLE2.PRT. The tilde

(~) delimiter is required.
To revoke definition of a keyword use >-DEFINE @keyword. See also >SHOW_DEFINES to display a list of all defined keywords.

>Define can also be used to do simple math to set the value of other defines. For instance:

Example:

```
>define @TEXTSTART 7001
>define @LASTDATA = TEXTSTART - 1
>echo @LASTDATA "will echo 7000"
```

Note that the "=" sign after the define. You can use other defines and any math statements in the defined variable (+. -, /, *).

Another option is to use two (@@)at signs:

Syntax:

```
>DEF @@name string
```

This stores the string in a DB file (must be opened ReadWrite) with the specified name (14-character limit).
By default, defined items are not expanded when they appear inside a quoted string. Use >FILL_DEFINES_INSIDE_QUOTES to have define items expand.

See also >DO_ALL, >IF_DEFINED, and >SHOW_DEFINES.

>DELETE

Deletes a file. This overrides any operating system level DELETE command. See also >PURGE. If you want to use your operating system's version of the delete command, use >SYSTEM DEL (See the >SYSTEM command.)

Syntax:

```
>DEL file
```

>DEMO

Suppresses certain program messages and all messages associated with ampersand (&) and meta (>) commands. This can be used in applications to keep program messages to the user at a minimum.

>DISK_ROOM

Displays the number of bytes available on the current drive. You can specify the actual number of bytes that must be available before a program can be run. In addition, you can redirect the program to another drive if the specified number of bytes is not available on the current drive. You can set this globally for all programs in the INITIAL file located in CFMC\CONTROL.

This meta command applies only to DOS-based systems.

Syntax:

```
>DISKROOM bytes >drive:
```

Examples:

```
>DISKROOM
```

Displays “Room on current drive: n bytes”

```
>DISKROOM  10000000
```

Checks for this number of available bytes on the current drive. If not available the program will terminate with: (ERROR #4129) Only n bytes available on disk, but n bytes needed now.

```
>DISKROOM 10000000 >D:
```

Checks for 10MB on the current drive and if not available will then try running the program on the D drive.

>DO_ALL

Used with >DEFINE, allows you to define a keyword without an @ sign, but the keyword will be case sensitive. >-DO_ALL is the default, meaning that keywords must be defined with an @ sign and case is ignored. In order to be effective, you must enter >DO_ALL before >DEFINE.

Syntax:

```
>DOALL
```

>DOS

See >CALL_DOS

>DUMP

Sets the switches which will display information useful for programmers or turn on special features. Dump switches will continue to display until they are turned off.

Syntax:

```
>DUMP letter#
```

Dump switches are case sensitive. You can turn several switches on in one command.

Example:

```
>DUMP S1g2
```

Use a minus sign to turn off a dump switch.

Example:

```
>DUMP -g2
```

You can also turn on or off all of the switches set with a particular letter.

Syntax:

```
>DUMP letter
```

Example:

```
>DUMP S
```

Use this syntax with extreme caution since this will turn on several switches at once and may cause unexpected results.
Use the SHOW option to see which dump switches are set.

Example:

```
>DUMP SHOW
```

>ECHO

Echoes text from a spec file back to the console. You could insert messages at various points in your specs so that when you execute them with PREPARE or Mentor, you'll know how far the program has run. This is particularly useful when your run results are being sent to a disk file and you would like to have an indication of how far the program has progressed. This overrides the operating system's >ECHO commands. Use `>SYSTEM echo` for the operating system version of the command.

Syntax:

```
>ECHO text
```

Example:

```
>ECHO Finished with section 1.
```

>ECHO_DEFINES

Shows the assignment of the defined keywords in the list file.

Syntax:

```
>ECHODEF
```

Example:

The following lines in a spec file:

```
>ECHO_DEFINES
>DEFINE @STUDY Gemini
>PRINT_FILE @STUDY
>USE_DB @STUDY
```

would generate these lines in a list file:

```
>DEFINE @STUDY Gemini
>PRINT_FILE @STUDY
(after define) >PRINT_FILE Gemini
>USE_DB @STUDY
(after define) >USE_DB Gemini
```

The default is >-ECHO_DEFINE.

>EDIT

Allows you to edit the text of a DB entry, i.e., title, banner, stub, while in edit mode. The item must be limited to 23 lines (screen limit). Items exceeding the 23 line screen limit can be edited in an outside program editor (see >DB_TO_FILE).

Syntax:

```
>EDIT newname dbentryname
```

This will save the edited item with the label newname. Or, you can use either newname or dbentryname alone:

>EDIT dbentryname Stores the edited item in the DB file with the same name (writes over the old item).

>EDIT newname Presents a blank full screen editor, andsaves the edited information as newname.
Press ESC to exit edit mode.

You must have a DB file open with ReadWrite access to store the DB entry. See >USE_DB and >CREATE_DB DUPLICATE=Warn, Quiet, or Replace if you want to replace items.
See also >EDIT_PREVIOUS.

>EDIT_FILE

Allows you to edit of an existing ASCII file. The file must be 23 lines or smaller.

Syntax:

```
>EDITF filename
```

>EDIT_FILE does not create a new file; your changes are saved to the existing file.
This will edit small ASCII files with CfMC's editor, or if you use SET EDITOR= *your_editor_name* (before running the program) it will use your editor to edit the file.

Syntax:

```
SET EDITOR=path\executable_file_name
```

Example:

```
SET EDITOR=C:\NORTON\NE.COM
```

Press ESC to exit edit mode. See also >DB_TO_FILE.

>EDIT_PREVIOUS

Allows you to edit the line you just entered. In interactive mode, the line is re-executed by the program once you press ESC to stop editing. >-EDIT_PREVIOUS will pass the previous line back as if retyped.

Syntax:

```
>EP
```

>ELSE Controls specfile execution; specified after a >IF_DEFINED. This allows you to specify actions to occur if the >IF_DEFINED condition is not true. You can specify the @keyword on the >ELSE line to be sure you are in the right

nested block of the >IF_DEFINED.

Syntax:

```
>ELSE
```

Example:

```
>DEFINE @a command
>IF_DEFINED @a
specs or another meta command
>ELSE @a
specs or another meta command
>END_IF
```

>ELSE_IF

Allows you to specify another defined @keyword within a >IF_DEFINED @keyword block as a condition for executing what comes next.

Syntax:

```
>ELSEIF @keyword
```

>END_IF Ends the IF block begun with >IF_DEFINED. To ensure that you are ending nested IFs correctly you can specify the @keyword specified on >IF_DEFINED on the >END_IF. If the keywords do not match you will get an error. (ENDIF)

Syntax:

```
>ENDIF <@keyword>
```

See also >ELSE.

>END_OF_FILE

Causes an end-of-file on the currently opened spec file. Use this command to indicate the end of a >FILE_TO_DB item.

Syntax:

```
>EOF
```

>END_REPEAT

Required at the end of each REPEAT block.

Syntax:

```
>ENDREP
```

See also >REPEAT.

>-ERROR_LINE_NUMBER

Suppresses the errors which indicate which line in the spec file an error occurred. These errors can be confusing and are not accurate if you have referenced another file with an ampersand, and so you can choose to shut the errors off. MPE Users: This option is especially useful if your input spec file is a QEDIT or HPEDIT file containing line numbers, because Mentor will include these line numbers, and the makes the error line numbers redundant.

Syntax:

```
>-ERRLNNUM
```

>FAKE_TIME

Allows you to set the system clock for CfMC programs. This is useful to set the same run date/times on output printed at different times (for instance, if you are checking for differences between output files over time), or to test a phone study questionnaire by setting different times and seeing which numbers are coming up using the FONESIM utility.

>FAKE_TIME reports the time it sets the system clock to when you use it.

NOTE: *Check this carefully. >FAKETIME never generates an error for bad syntax. It leaves the module with the illegal value unchanged.*

Syntax:

```
>FAKE_TIME <day month year time>
```

Example:

```
>FAKE_TIME 01 AUG 99 11:15
```

Options:

day The day of the month field must contain a number that could occur in the month specified.

month Only the first three letters of the month field are recognized. AUG, August, and AugXXX will all set the month to August.

year Legal values for the year are two digits for 1980-1999 and four digits for 2000-2027.

time Use 24 hour clock time, between 00:00 to 23:59 (use 00:00 for mid-night, not 24:00). The colon (:) is optional.

>FAKE_TIME without any options returns you to actual system time. The >FAKE_TIME command controls the time and date information that will be displayed in a CfMC program run. The >FAKE_TIME setting can be changed during the course of a run.

The >FAKE_TIME command affects the Mentor System variable, DATE_TIME. It will also affect the results of >STOP_WATCH commands. Once a >FAKE_TIME is set, >STOP_WATCH will only report 0 minutes and 0 seconds in its total time field, unless a >FAKE_TIME command with no arguments is encountered. When >FAKE_TIME has been reset to the actual time, the total time reported by >STOP_WATCH will be more or less correct.

The >STOP_WATCH command will report the difference between two FAKETIMEs in its second field. When the difference between two FAKETIMEs is a negative number, 0 minutes 0 seconds is returned.

>FILE_TO_DB

Writes a file with variable specifications to the DB file (opened with ReadWrite access).

To put specs in a DB file under a ‘name’:

Syntax:

```
>FILETODB name filename
>FILETODB name #
```

A pound sign (#) after the name means to read from the spec file immediately following (or the console) until an end-of-file is encountered.

Example:

```
>FILETODB mystub #
TOTAL
NO ANSWER
[print_row=AR] ANY RESPONSE}
>EOF
```

This example would store these lines in the ‘save to’ DB file under the name MYSTUB. This item could then be referenced to build tables with this label set as the tables’ stub preface. See also >DB_TO_FILE for the reverse procedure.

>FILL_DEFINES_INSIDE_QUOTES

Expands @define items when they appear inside a quoted string. The default is that they are not expanded.

Syntax:

```
>FILLDEF
```

Example:

```
>DEFINE @month March
PRINT "This is the report for @month"
```

would print:

```
This is the report for March.
```

See also >c v‘DEFINE, >ECHO_DEFINES.

>FORCE_HARDCODE

You can put this command in the “mentinit” file and all compules will create QSP files with “hard-coded” (assigned) data locations. This works the same way as the compiler directive (!hardcode), but it does not have to be specified separately for every questionnaire.

>HALT

Controls when the program stops and waits for Enter to continue. This will work when the program is expecting possible additional input from the console/keyboard (i.e., almost all cases except when your program command included your spec file name without an ampersand preceding it).

Syntax:

```
>HALT option
```

Options:

ANY Stops the program at any program message.

ERROR Stops the program at errors if console available (&specfile).

NONE Disables other >HALT commands. This is the default.

WARNING Stops the program at any error or warning message if console available (&specfile). If you enter **HALT** with no options, it stops the program.

>HELP

Displays the main help menu or help for the current program block.

Syntax:

```
>HELP
```

After loading either the Mentor or the PREPARE program, typing >HELP at the first prompt will display the main help menu. To get help on an item displayed on this menu, type the name of the item and Enter. If you are unsure of what to type, just type any letter and Enter. This will display list of available help keywords (options).

To move to additional screens, enter MORE and press Enter. Back up to a previous screen by typing the caret (^) on your keyboard. You can always return to the main help menu by entering MAIN and pressing Enter. To exit help and return to the program prompt, press Enter.

From many help screens you can get a complete list of the keywords available for that command, such as ~DEFINE EDIT= or STUB=. When the list is long it will stop after one screen, and allow you to page back and forth through the listing with the minus (-) and plus (+) keys.

>IF_DEFINED

Says that if the @keyword is defined, then do what comes next. Requires >END_IF to end the IF block, and allows >ELSE to do actions if the statement is not defined. Use this as a conditional statement to control branching in a spec file.

Syntax:

```
>IFDEF @keyword
```

See also >ELSE, >ELSE_IF and the IF/ELSE/ENDIF commands in MPE, DOS, and UNIX operating systems.

>JULIAN_YEAR_LENGTH

Sets the length for the Julian Year field of a date variable. Legal values are “only 2” and 4. By default, the width of the Julian Year field is 4 digits. Using this command with no value returns it to the default.

Syntax:

```
>JULIANYEAR=4
```

>KEY_DELAY

Sets the speed of keystroke display (in milliseconds) for replaying a >SAVE_KEYS file.

Syntax:

```
>KEYDELAY # &filename.KEY
```

The higher the #, the slower the rate of replay. The KEY_DELAY command is optional. By not specifying KEY_DELAY you allow the key file to replay at maximum speed but you will not be able to follow the keystrokes. 100 is a reasonable speed.

There are several # values that do special tasks:

>KEY_DELAY -1 Requires that a key be pressed at each word, Enter, ESC, or F - # key to continue the display.

>KEY_DELAY -2 Requires that a key be pressed at each keystroke to continue the display.

>KEY_DELAY #, # The first number controls the replay speed of text and the second number controls the replay speed of all other keystrokes (e.g.,: Enter, ESC, cursor arrow keys, function (F) keys).

This option is very useful for demonstration purposes, since text can be played back to the screen quickly while other "operational" keystrokes can be slowed down enough for the observer to see exactly how something was executed in that particular program.

See also >SAVE_KEYS.

>LIST_DB

Lists entries in a DB file; the default is to list all entries. Groups of items can be selected by variable type or by specifying a mnemonic pattern. If more than one DB file is open, you must specify which to list. The listing can be sent to the console and/or an ASCII file. Use >TERMINAL-PAUSE=# to pause after nth line prints to the screen.

You will get a message if no items are listed.
LIST_DB and LIST_DB_CONTENTS do the same thing.

Syntax:

```
>LISTDB
dbname,listname,SORT=sort_option,APPEND,TYPE=type,PAT=(pattern),TEMP="text"
```

Options:

dbname Is any open DB file, and is not required if only one DB file is open. If not specified, a comma must be entered as a place holder if anything follows it on the line.

listname Name of the ASCII file which will contain the listing. When a list file is specified the file will be given the extension DCL. A plus sign (+) in front of the listname will send the listing to the console *and* to the list file. If left off, a comma must be entered as a place holder if anything follows it on the line; output will go to the console only.

SORT= Specifies how to sort the DB file and which items to display either to the console and/or DB list file.

Syntax:

```
SORT=sort_option
```

Sort_options:

LOCATION Lists items in the order they were created in.

NAME Lists all items sorted by name (default). It is not necessary to specify SORT=NAME, just >LISTDB dbname.

QQNUM Lists only items that have a question number, meaning items that were made by the PREPARE program or ~DEFINE PREPARE=.
The list is sorted on the qqnum. You can further qualify this sort by specifying a specific range after -QQNUM.

-QQNUM causes the sort to include items that do not have a question number. These are items made in Mentor such as data variables or table definitions.

Example:

```
>LISTDB rrunr,+rrunr,SORT=QQNUM(QN1-QN4)
```

In this example, QN1 is the beginning and lowest name and QN4 is the highest name that will be included in the list.

APPEND Append to the DCL file if it's there, otherwise append to a new file.

TYPE= States the specific variable types which should be listed.

Syntax:

```
TYPE= option
```

Options:

ALL Lists all items.

TABLES Lists all tables.

TABSETS Lists all table sets.

VARIABLES=ALL Lists all variables.

VARIABLES=# Lists specific variable types. More than one number can be used here. You can use the numbers 1-8, where:

```
1: VAR ($|$S)
2: Not used
3: FLD (=)
4: (#)
5: NUM
6: CAT (^)
7: TEX ($T)
8: ($P)
```

Example:

```
>LISTDB demo,catvars,TYPE=VAR=6
```

PATTERN= Specify a mnemonic pattern which the variable names must match to be included in the listing. The pattern may include alpha or numeric characters, and wildcards such as *, ?, or #.

Syntax:

```
PAT=(pattern1,pattern2, etc.)
```

The patterns may be separated by a comma or space.

Examples:

```
>LISTDB work,,PAT=(te*)
>LISTDB demo,,PAT=(QQ*, a*, te??????)
```

TEMPLATE= May state text to be listed along with each variable in the list. he template may be extended beyond one line by closing the quotes and using & at the end of the first line; the maximum length of a template string is 300 characters. An exclamation point (!) in the template is the place holder for the variable name, and a vertical bar (|) will start a new line.

Lines generated with the TEMPLATE option cannot exceed 132 characters. Use

line breaks (\\) to break long lines at 132.

Syntax:

```
TEMP="text ! and|a new line "!" here" &&
",which can be extended."
```

Examples:

```
>LISTEB demo,+listoe,TYPE=VAR=17, &&
TEMP="~SHOW !|"
>LISTDB demo,cats,TYPE=VAR=6,TEMP="'CAT"&&
" variable: !|~SHOW !"
```

The list file can be edited and used for getting multiple entries from a DB file without specifying them individually. This can be done by typing && before each name and then using &filename.DCL to bring in the file during a PREPARE run.

Example:

```
>LISTDB RRUNR,+RRUNR
```

This example lists the contents of the DB file RRUNR to the screen and to the file RRUNR.DCL (DOS and UNIX or RRUNRDCL in MPE). The default sort lists all items alphabetically.

Here is a sample DB file list that includes examples of different DB variables:

```
AXIS1 '' type = 3, g1=0000, g2= 8, g3= 0, ver = 1
BAN1 '' type = 123, g1=0104, g2= 0, g3= 132, ver = 1
EDT1 '' type = 123, g1=0007, g2= 0, g3= 132, ver = 1
HEAD1 '' type = 123, g1=0104, g2= 0, g3= 132, ver = 1
PROC1 '' type = 124, g1=0001, g2= 8, g3= 0, ver = 1
QQ001.00 '' type = 1, g1=0015, g2= 15, g3= 100, ver = 1
QQ002.00 '' type = 1, g1=0023, g2= 15, g3= 200, ver = 1
QQ003.00 '' type = 1, g1=0023, g2= 15, g3= 300, ver = 1
QQ004.00 '' type = 1, g1=0026, g2= 15, g3= 400, ver = 1
ROW1 '' type = 1, g1=0024, g2= 15, g3= 0, ver = 1
STUB1 '' type = 123, g1=0005, g2= 1, g3= 132, ver = 1
TABSET1 '' type = 123, g1=0009, g2= 0, g3= 132, ver = 1
T001 '' type = 4, g1=0000, g2= 10, g3= 0, ver = 1
```

Here are the elements of the dbfile and their meanings. Let’s look at one specific

line of the example above:

```
QQ001.00 '' type = 1, g1=0015, g2= 15, g3= 100, ver= 1
QQ001.00 The name of this DB item.
```

type = 1 The DB item type. It can be:

1	variable
3	expression
4	table
7	print statement
8	stats statement
123	specs (meaning any text variable or >FILE_TO_DB item)
124	procedure
324	Search_control
555	rank control
635	Read_control

g1=0015 information provided in g1 varies depending on the type of item. The number in g1 is a hexadecimal number, individually referred to as the 1000s place, 100s place, 10s place and the fourth place is the unit.
The third and fourth places in g1 indicate the variable type for type= 1 DB items. The third or 10s place can be:

0	Boolean
1	Number
2	Cats
3	Vector
4	String ($T,P,or $)
5	Region__!

The fourth place or unit position can be:

1	String ($)
4	# (Numeric range)
5	Numeric
6	CAT (^)
7	TEX ($T)
8	Punch ($P)
9	[? (EZW) "made with EZWriter
A	^ optimized

The fourth place will contain an “A” if a type 6 (^) variable was converted to an optimized variable for faster execution. This is an internal program routine.

For type= 123 variables the fourth place indicates variable type for text variables (or items that come >FILE_TO_DB):

4 Banner= or Lines=
5 Stub=
8 Print statement
9 Table_set=

When the fourth place is 4, then the 1000s and 100s place will indicate the number of lines in that item.

Example:

```
BAN1 '' type =123, g1=0104, g2=0, g3=132, ver = 1
```

In this example the 1 in the 100s place in g1=0104 indicates this item contains one line. For type= 3 (expression) g1 will be all 0s, i.e., g1=0000 meaning no variable type.

g2= 15 g2 usually indicates the software version for this type of item. This number changes for a particular variable type only when there is a CfMC program change or enhancement to the internal routine that makes that type of variable. A programming change could cause a version conflict for a specific variable type. Note: This is *not* the same as the DB file version. Program version changes can cause an old DB file to have a version conflict when a newer version of the program tries to access it. If the DB item is a stub (e.g., type=123 g1=0005) then the numbers after g2= indicate the number of lines in the stub.

g3= 100 Information in g3 varies depending on the type of DB item:
If g3= 0 then there is no further information for this type of DB item.

For type= 1 variables this is the internal question number of an item made with either the PREPARE program or ~DEFINE PREPARE=.

For type= 123 variables this indicates the record length of the file and no line in the file will be longer than this number. This is mostly significant for HP users.

ver = 1 The number of times the name of this item appears in the program’s internal DB directory. When you edit a DB item with either DUPLIATE=QUIET or WARN set on the ‘save to’ DB file, then this number will increment. The

DBUTIL option REVEAL will show all versions of a particular DB item. DUPLICATE=REPLACE will not increment ver =.

>LISTFILE <filename> option1

The listfile command is used to switch to a new list file from within a spec file. The original list file will be the list file defined on the command line. This can be used with the same options as the listfile: command-line keyword.

Syntax:

```
>listfile list1
```

(This will be directed to the list file named "list1")

```
>listfile list2,echo
```

(This will be directed to the list file named list2 and it will echo to the screen)

```
>listfile list1 append
```

(This will be appended to the list file named "list1")

See Appendix D: Command Line Keywords, for more details and a description of all the options.

>LOCATION_FORMAT

Allows the user to specify the format of data locations in Script Composer, the QSP comment line, CHK, HRD and SUM file data locations, data locations in HOLE tables, CLEANIT displays, and error and warning messages. You can enter data locations in any of the three formats, but output will be formatted according to the specification in this command.

Syntax:

```
>LOCFORM option
```

Options:

1	Displays data locations as absolute columns (i.e., 113).
1/1	Displays data locations as record/column (default) (i.e., 2/33).
A01	Displays data locations as letterdigitdigit (i.e., B33).

Example:

```
>LOCFORM 1/1
```

You can specify >LOCFORM in the file INITIAL in CFMC\CONTROL (DOS or

UNIX; CONTROL.CFMC in MPE). This will globally control location format for any CfMC program.

>MULTIBYTE

Reads ASCII characters 0-255 from a data file. Without this command, ASCII characters in a data file below 32 or above 126 are converted to a space. You can use this command when your data file contains foreign characters or if you are using a printfile as a data file and it contains graphical line characters. Use >CHARACTER_SET=ASCII/EXTENDED_ASCII if you want to read additional ASCII characters in a spec file.

>NUMBER_ADJUSTMENT=

Controls truncation in the addition of binary real numbers. For example, the sum of a set of weights may be 340.5 in decimal, but 340.499999999998656 binary, which rounds to 340, not 341.

Syntax:

```
>NUMADJ=#
```

The number you specify on this command will be added (or subtracted if negative) to every number printed. The default is .0000001.

Syntax:

```
>NUMADJ=*
```

Specifying * as the adjustment will tell the program to return to the default adjustment.

>NUMBER_OF_FILES

DOS ONLY: Adjusts the number of CfMC files you can have open at once. The default is 100 files. This means you can have about 90 input files in a typical run. If you use >NUMBER_OR_FILES to raise the limit, you must adjust the DOS FILES command in your config.sys file as well.

>PASS_comments

This directs whether or not you want to pass “comments” from user spec files to the .qsp file when compiling questionnaires. The default is NO. This is put in the MENTINIT file. Comments made using the “comment syntax” (!comment xxxxx) are always passed when this is set to YES.

>PAUSE=n

Pauses after every n lines prints to the terminal. (PAUSE)

>PRINT_FILE

Creates an ASCII file for tables or output from PRINT commands. These files have an extension of PRT.

Syntax:

```
>PRTF filename,#n,afilename=,prtoptions
```

Options:

filename The name of the print file. You may also specify the filename as NULL or $NULL meaning the file will not be saved to disk. If you use $filename, you do not need to include the PRT extension with the filename.

Example:

```
>PRT NULL #1
```

In MPE, you may specify filename as *filename to back reference a file equation. See the example under *List File* in *Appendix D: CfMC* CONVENTIONS.

#n The number of the print file. The default is #1, and the allowable numbers are 1-99. You can print to multiple files in one run this way, but you can only have one print file ON (see below) at one time. There is no effective limit on the number of print files that can be open at one time. The number depends on the operating system you are running. [DOS USERS: the FILES= statement in CONFIG.SYS limits the total number of files that can be open simultaneously (including operating system and program files; use >STATUS FILES to list all open files). When you exceed this limit the program will print an error message. It is possible to increase the FILES= specification in the CONFIG.SYS file to allow more print files to be open, but this can cause other errors if system memory is insufficient.]

afilename= is the alternate or alias name for this print file. You would specify this instead of a #n after the filename. Names can be alphanumeric but must start with an alpha character. See examples below.

prt options:

APPEND Adds the tables to the end of the file if it already exists, otherwise creates the file.

BOTTOM_MARGIN=# Number of blank lines at bottom of page. Default is 3. This affects the number of lines that will print on the body of the page. The body

of the page is the PAGE_LENGTH minus the TOP and BOTTOM margins. You will increase or decrease the number of lines in the body of the page by changing the default for this option. The body of the page must have at least five usable lines remaining after the top and bottom margins are specified.

COPIES=n In MPE, this specifies number of times to print the spool file. ECHO Prints the table to the print file and to the screen.

FIXED_FORMAT Gives each line in the print file a fixed length, for example, each line will be the defined maximum line length(default 132) filled with blanks. Use this option if you want to use your print file as input for a Mentor run (MPE only).

FORM_FEED Specifying -FORM_FEED lets you override program default form feeds (^L).

FORMS= "forms message" In MPE, this specifies the forms message for spool files.

HEADER_PAGE Prints a leading header page with the following information: print file name, program, version, mode, file names, date and time.

Example:

```
Header Page for CfMC Printfile c:\tests\example.prt
PROGRAM Mentor 7.1 (1,08May97) with input/output:
test.spx to -test.lfl
11 JUL 1997 14:50
```

You can specify this option in either the INITIAL or MENTINIT file to make it the default for all sessions, as follows:

```
>PRT; HEADER_PAGE.
```

LASER_CONTROL=<filename> Refers to the name of the laser control file (located in the CfMC CONTROL directory or group). This file contains the escape sequences that will be passed to the print file when back slash (\) commands for text enhancements such bold (\B) or underlining (\U) are encountered. You may have more than one laser control file.

LASER_NUMBER=# Specifies which laser printer to use. See *Appendix D: CfMC CONVENTION*S, *Command Line Keywords* for more information on LASER_CONTROL and LASER_NUMBER.

PAGE_LENGTH=# Total number of print lines on a page. Default is 66.

PAGE_WIDTH=# Number of characters per print line. Default is 132. Maxi-

mum is unlimited.

TOP_MARGIN=# Number of blank lines at top of page. Default is 3. See BOTTOM_MARGIN. You may set this to zero (0).

USER Allows you to override the PRT extension, and specify any extension you wish.

Syntax:

```
>PRTF filename^ext,USER
```

If you do change the print file extension then you must refer to the new extension (filename^ext) if you reissue the >PRTF statement (see >PRTF ON and OFF).

Two options can be used as a toggle switch:

1 ON

Makes this file the default print file (once the file has been opened with a previous >PRTF command).

Syntax:

```
>PRTF filename ON
```

2 OFF

Turns off printing to the default print file (previously turned on with >PRTF ON). Does not close any print files, but you must specify the file to print to after turning the default print file off.

Syntax:

```
>PRTF filename OFF
```

You can also have the name assigned by a >STUDY_NAME command:

Example:

```
>STUDY BANK
>PRTF *
```

This >PRINT_FILE command will substitute the name specified on the >STUDY_NAME command (BANK) for asterisk (*).

You can use >PRINT_FILE with no options:

Example:

```
>PRTF;
```

This example will close a print file and will not open another. If you have more than one print file open, you must specify a file name before the semicolon (;).

This is a simple example of how to print to two print files at once:

```
>PRTF file1 #1
>PRTF file2 filename=abc
>PRTF file1 ON
~CLEANER
PRINT "this prints to file1, the default printfile"
PRINT #1 "this also prints to file1"
PRINT #"abc" "this prints to file2"
MODIFY fname[1.4$]="abc"
PRINT #fname "this also prints to file2"
```

You must close a print file to print or browse it.(>PRN, >BROWSE).
See also: ~DEFINE EDIT= and ~SET BOTTOM/TOP_MARGIN, PAGE_LENGTH/WIDTH.

>PRINT_FILE_DEFAULTS

Allows you to set a group of commonly used print file options that will be used each time the >PRINT_FILE command is issued in that file. This is often put in the mentinit file to set standard print controls.
>-PRINT_FILE_DEFAULTS will turn off the group of print options.
Syntax: `>PRTFD option option option`
You can set options with >PRINT_FILE_DEFAULTS and then modify them with the >PRINT_FILE command.

Example:

```
>PRINT_FILE_DEFAULTS page_length=60 page_width=80
>PRINT_FILE file1
~CLEAN PRINT "file1 has a page length of 60 and a
width of 80."
>PRINT_FILE file2 page_width 20
~CLEAN PRINT "file2 has a page length of 60 and a
width of 20."
>-PRINT_FILE_DEFAULTS
>PRINT_FILE file3
~CLEAN PRINT "file3 has the system defaults of a page
length of 66 and a width of 132"
```

See also >PRINT_FILE.[ToolBar-Bar7]BarID=594

>PRINT_REPEAT

As a repeat sequence (>REPEAT) is expanded in the program it will be printed to a list file as it is generated (or it will be shown on screen if no list file is specified). This lets you see how the program is treating the repeat. This is the default and need not be specified unless you need to turn off >-PRINT_REPEAT, which suppresses the listing of the expanded repeat sequence.

Syntax:

```
>PRTREP
```

>PRN

Checks for an active printer port, and if one is found prints the file named to that printer. (DOS only)

Syntax:

```
>PRN filename
```

Only print files which have been closed can be printed (see >PRINT_FILE).

>PURGE_empty_tr(files)

If a TR file is built and no records are added to it, this controls whether or not you want to keep or delete the files when the job is done. The default is to keep the files.

>PURGE_empty_ascii(files)

Similar to the meta command above: If an ASCII file is built and no records are added to it, this controls whether or not you want to keep or delete the files when the job is done. The default is to delete the files.

>PURGE

This command deletes a file. This overrides any operating system level PURGE command. Use >SYSTEM purge to use the operations system version of the purge command.

Syntax:

```
>PURGE file
```

See also >DELETE, >SYSTEM

>PURGE_SAME

Purges any existing file (except the list file) that has the same name as any new file created by the program. Without PURGE_SAME, the program renames old files by changing the filename's first letter to the next alpha character. (AFILE, for

example, would become BFILE.) You need to specify this before the program creates the new file.

Syntax:

```
>PURGESAME
```

Use >-PURGE_SAME to return to having files renamed. If you want >PURGE_SAME to be a default for all of your runs, put in your init or mentinit file.

>PUT_CHARACTERS=

Allows you to specify three characters which designate how to fill a field containing zeros (either integer or floating point) or missing data.

Syntax:

```
>PUTCHAR=abcd
```

The first character specifies the field filler for integer zeros, the second is for floating point zeros, and the third character is for missing data. A fourth character may be specified for empty cells. The default value for >PUT_CHARACTERS is "-" for integer zeros, "0.00" (to the appropriate decimal significance) for floating point zeros (indicated as "0"), and "?" for missing data values; to return to these defaults, specify >PUTCHAR=-0?.

Any character (except for B or Z) can be printed in the rightmost column of all appropriate fields by entering that character in the location corresponding to the type of field. Put B in any of the fields to indicate blanks, or Z for zeros.

If you have more than one PUTCHAR set (from >PUT_CHARACTERS, ~DEFINE_EDIT=PUT_CHARACTERS, or ~DEFINE_COLUMN_INFO=PUT_CHARACTERS), the >PUTCHAR values are read first, then individual tables may override that with an EDIT=PUT_CHARACTERS, and individual columns or rows in a table may override that with COLUMN_INFO=PUT_CHARACTERS or STUB=[PUT_CHARACTERS] respectively.

>QUEUE=

Says which queue to run C-Mentor in under the MPE operating system.

Syntax:

```
>QUEUE letter/=?
```

Options:

letter Can be B, C, D, or E, the default is C.

=? Shows the current queue. (Syntax: >QUEUE=?)

>-QUEUE Turns the current queue off.

>QUIT

Immediately quits the current program and returns to the operating system prompt. Temporary files used by the program are not purged. Use this command with caution because changes made to open data files will not be saved.

Syntax:

```
>QUIT
```

If there is any text on the line after the QUIT, that text will be displayed by the program when it terminates.

Syntax:

```
>QUIT ERRORS=#
```

The job will quit if the number of errors up to this point is greater than or equal to the number specified here (#). The default is 200.

>RANDOM_SEED=

Allows you to set a specific seed for the Mentor vector function RANDOM_CATEGORY, and the Survent function RANDOM.
This way, you can always generate the same results for these functions.

Syntax:

```
>RANDOMSEED=#
```

>READ

Allows the user to enter one command from the console (including Enter) before the program returns to processing. The user will see this prompt: ===> This command can only be used on an interactive run (e.g., Mentor &spec.spx con)

Syntax:

```
>READ
```

Example:

```
~INPUT data.asc SELECT=&
>READ
~OUTPUT subset.asc, WRITENOW
~END
```

>RENAME

Renames a file. This overrides any operating system level RENAME command.

Syntax:

```
>REN file1, file2
```

>REPEAT

Sets up a repeating pattern to produce multiples of specifications, using variable names for changing items within the repeated sequence. The repeat elements are then used in the specifications (which must be between >REPEAT and >END_REPEAT commands). The program will repeat the specifications as many times as needed, based on the number of repeat elements, up to 1000 iterations. The maximum line width that can be generated by >REPEAT has a default of 132 characters. This can be changed by the command line keyword SPEC_WIDTH (see *Appendix D* for a description of SPEC_WIDTH).

Syntax:

```
>REP $NameA=item1a,item1b,item1c;$NameB=itemna
,itemnb,itemnc; STRIP="literal"
```

**$Name=** The repeat elements (limited to 19 alphanumeric characters) must be preceded by a dollar sign ($). They are followed by an equal sign and the specific variables.

item1a Variables must be enclosed in quotes if they contain spaces, commas, or semicolons, or if they don't begin with a letter or numeric digit. Repeat elements are not case sensitive.

Example:

```
>REPEAT $A=1,2,3;$B=Apples, Oranges, Bananas
{RATE$A:
How would you rate $B?
!CAT
3 Excellent
2 Fair
1 Poor }
>END_REPEAT
```

This example would generate three different questions, the first one being:

```
RATE1:
How would you rate Apples?
3 Excellent
2 Fair
1 Poor
```

Any dollar signs needed as part of the specification that precede a letter (e.g., $M) must be preceded by an extra dollar sign (e.g., $$M). Single dollar signs in other places are fine.

To return a quote, use two quotes in a row.

Example:

```
>REPEAT $A="The title is ""My Fair Lady!""
```

Repeat elements must end with an underscore (_) if the next character runs into the element name (i.e., $A_01 so the program looks for $A, not $A01. See REPEAT_VARS_ALPHA_ONLY.)

Example:

```
>REP $A=1,2,3;$B=30,31,32
VAR$A_A: [$B^1//4]
>ENDREP
Would generate the following:
VAR1A: [30^1//4]
VAR2A: [31^1//4]
VAR3A: [32^1//4]
```

You can also use an ellipsis (. . .) to designate a pattern in the variables. 10, . . . ,15 means 10,11,12,13,14,15, while 10,12, . . . ,20 means 10,12,14,16,18,20. The pattern can decrement as well as increment. The default pattern is 1; you can establish a different pattern by specifying two single variables before the ellipsis. Text cannot be set up in an ellipsis other than simple uses (i.e., a, . . . ,g).

Example:

```
VAR15: &
>REP $A=19,...,22;STRIP="WITH &"
[$A^1//5] WITH &
>ENDREP
Would generate:
VAR15: &
[19^1//5] WITH &
[20^1//5] WITH &
[21^1//5] WITH &
[22^1//5]
```

STRIP= lets you specify a string or literal that will be removed from the final element in the repeat string.

By default, Mentor replaces >REPEAT variables with their values as soon as it encounters them. >REPEAT can “pass through” variables as variables when >REPEAT is used in conjunction with >ALLOW_INDENT. When >ALLOW_INDENT is ON and file name with an ampersand within a >REPEAT

is indented, variables are passed through rather than expanded. All variables will be passed through, including $text strings. In this example, Mentor will not bring in the contents of the referenced file:

```
>ALLOW INDENT
>REPEAT $F=1,2,3
&file$F
>ENDREPEAT
```

Mentor will pass through the variables in file1, file2 and file3 and issue a warning:

```
**gen** &file1
(WARN #4275) AMPERSAND file &file1 being passed thru
**gen** &file2
(WARN#4275) AMPERSAND file &file2 being passed thru
**gen** &file3
(WARN #4275) AMPERSAND file &file3 being passed thru
```

See also >ALLOW_INDENT, >END_REPEAT, >PRINT_REPEAT and >REPEAT_VARS_ALPHA_ONLY.

>REPEAT_VARS_ALPHA_ONLY

Has the program read repeat element names as alphabetic only. Use this command before doing repeats so that when the program sees '$A01' in a spec line it knows that the '01' is not part of the repeat name. Otherwise you need to write it as '$A_01'.

Syntax:

```
>REPVARA
```

>RESET_DB

Closes all DB files currently opened with either a >CREATE_DB or >USE_DB command.

Syntax:

```
>RESETDB
```

See also >CLOSE_DB.

>RUN_LABEL=

Specifies text to print (approximately 50 characters including spaces) in place of the spec file name printed in the program information at the end of the run.

Syntax:

```
>RUNLAB=text
```

Example:

```
>RUNLAB=SAMPLE TABLES
Mentor 12May96 running SAMPLE TABLES at 27 MAY 1996 08:11.
total time: 0 minutes 17 seconds
```

>SAVE_AS_DB

Saves input entered prior to the command as one DB entry in the DB file opened ReadWrite.

Syntax:

```
>SAVEASDB dbname
```

See also >DB_TO_FILE and ~MAKE_ASQ.

>SAVE_AS_FILE

Saves all input entered prior to the command in a file of the specified name. This includes both what you type and commands called in from other files.

Syntax:

```
>SAVEASFILE filename
```

For example, you could use this command if you are working interactively in C-Mentor and have gotten several errors and you want to save what you have entered so far so that you could edit it. You could then use the edited file as your spec file.

You can also put this command in your init or mentinit file to have it create a file of commands each time you use Survent or Mentor.

>SAVE_KEYS

Saves all keystrokes after the command (up through the next >SAVE_KEYS, if any), in a file of the name specified. The file can then be called in (&file-name.KEY) and the program will rerun all keystrokes.

Syntax:

```
>SAVEKEY filename.KEY,#
```

Options:

filename.KEY The extension is optional; if left off, it will be added to the file-name. No other extension can be entered.

This saves and closes the file after n number of keystrokes is saved. Any number may be specified here. The default is to only save the keystrokes when the file is closed.

Example:

```
>SAVE_KEYS keeper,1
```

This will save and close the file after each keystroke. This is valuable when your system is crashing and you are losing the SAVE_KEYS file. This will slow response time considerably!

The key file can be edited, allowing you to delete or add key commands such as Halt which causes the keystroke display to stop at that point and wait for an Enter to continue. See also >KEY_DELAY.

>SHOW_CORE_FARMARK

Shows the current condition of core memory. This is useful if you are debugging problems related to lack of memory, such as CfMC GETCORE or CHECKCORE error messages. It shows you what program commands are using memory and how much memory remains. Your CfMC support representative can provide more information and assistance.

Syntax:

```
>SC
```

>SHOW_DEFINES

Lists keywords (created with >DEFINE) and their definitions.

Syntax:

```
>SHOWDEF option
```

Options:

***** Lists all keywords and definitions.

keyword Lists only the keyword specified. The at sign (@) must not be specified as part of the keyword.

letter*letter Lists all keywords beginning with the first letter and having the second letter somewhere in the name.

Example:

```
>SHOWDEF MENU
```

Also see >DEFINE.

>SHOW_KEY=

Shows the keywords (options) for the MSGFILE block number named on the command. The keywords are sorted in alphabetical order. The number (i.e., 14:) appearing at the beginning of each keyword is its position number in the list of keywords for that MSGFILE block.

Syntax:

```
>SK=####
```

Example:

```
>SK=11
44
Keywords from msgfile #1144 ( 16 entries):


14: Blank  B b(lank)
3: Column=  C= c(olumn)=
15: COLumn_STATistics_values  COLSTAT
col(umn)(_)stat(istic)(s)(_)(val)(ue)(s)
9: Cumulative_Percent  CP  c(umulative)(_)p(ercent)
10: FREQuency  FREQ  freq(uency)
4: Frequency_Decimals=  FD=  f(requency)(_)d(ecimal)(s)=
7: Horizontal_Percent  HP  h(orizontal)(_)p(ercent)
13: MINimum_Base=  MINB= min(imum)(_)b(ase)=
16: NUMber_FORMat= NUMFORM= num(ber)(_)form(at)=
5: Percent_Decimals=  PD= p(ercent)(_)d(ecimal)(s)=
11: Percent_Sign PS p(ercent)(_)s(ign)
12: PUT_characters=  PUT= put(_)(character)(s)=
2: Statistics_column S  s(tatistic)(s)(_)(column)
6: Statistics_Decimals= SD=  s(tatistic)(s)(_)d(ecimal)(s)=
8: Vertical_Percent  VP  v(ertical)(_)p(ercent)
1: Width=  W= w(idth)=
```

Here are some msgfile blocks commonly used:

COMMAND:	MSGFILE BLOCK NUMBER:
Meta Commands	4000
>CREATE_DB sizes options	2100
DB meta options	2101
>USEDB Options	2101
>CREATEDB Options	2103
DB DUPLICATE=Options	2104

COMMAND:	MSGFILE BLOCK NUMBER:
>LISTDB Options	3793
>LISTDB TYPE=Options	3794
Mentor Function Keywords	5061
~CLEANER keywords (non procedure)	2684
~DEFINE	
AXIS= $[keywords]	2583
EDIT= keywords	1216
COLUMN_INFO=options	1144
LINES= options	1107
STUB= [keywords]	1217
[PRINT_ROW=] options	1218
~EXC keywords	1213
~INPUT options	1210
~OUTPUT options	1211
~PREPARE	
header options	1446
compiler commands	1466
~SET keywords	1207
Reserved Keywords	1203

Reserved keywords are words that are reserved by Mentor. They cannot be used as variable names.

To get a listing of reserved keywords, enter >SK1203 and it will produce a listing

similar to this:

```
15: ALTer_FLAG ALTFLAG alt(er)_flag
10: CASE_ID CASEID case_id
20: CASE_NUMBER CASENUMBER case_number
21: CASE_WRITTEN CASEWRITTEN case_written
7: CATegorieS CATS cat(egorie)s(
29: CHecK_ERRor CHKERR ch(ec)k_err(or)
13: DATE_TIME DATETIME date_time
9: DELete_FLAG DELFLAG del(ete)_flag
5: DUDDUDdud
12: EOF_DATA EOFDATA  eof_data
8: ERRor_FLAG  ERRFLAG  err(or)_flag
25: ERRORS  ERRORS  errors
2: FALSE  FALSE  false
11: FIRST_CASE  FIRSTCASE  first_case
26: JULIAN_DATE JULIANDATE  julian_date
31: LAST_CASE  LASTCASE  last_case
22: LINE_NUMBER  LINENUMBER  line_number
27: LNNUM  LNNUM  lnnum
6: MATH_VALueS MATHVALS  math_val(ue)s
3: MISSING  MISSING  missing  NOT  NOT not
23: PAGE_NUMBER  PAGENUMBER  page_number
28: PGNUM  PGNUM  pgnum
16: RANDOM_VALue  RANDOMVAL random_val(ue)
24: TABle_NAME  TABNAME  tab(le)_name
30: TEXT_AREA_STATUS TEXTAREASTATUS  text_area_status
4: TOTAL TOTAL  total
1: TRUE  TRUE  true
14: VALueS  VALS  val(ue)s(
```

>STATUS

Displays the current status of the specified files.

Syntax:

>STATUS <option>

Options:

ALL Displays the status of all files currently opened. This is the default and can also be done by entering >STATUS with no option.

DB Displays the status of your DB file(s).

FILES Displays a list of all open files including msgfile, console, temp files, input, and output files. It also lists how many blocks of far memory are available.

INPUT Displays the status of your input file, and lists the study code (if any).

OUTPUT Displays the status of your output file, and lists the study code (if any).

An example of >STATUS output follows.

Example:

```
~INPUT DATA1
~OUTPUT DATA2
>STATUS
c->studycode: ()
TEMPNAME = te205002 TEMPVARNAME = TV000001
spec_wid=132, bufsizes: spx=2048, prt=2048, tr=2048, bf=4096, db=2048,
big=4096, misc=2048
INPUTfile, study_name=rrunr, 1 cases read
DOT every 10 cases
total case is 640 columns: data in 1.640, 640 work columns
modes: allow_update-N, allow_new-N, use_deleted-N
Print case ID when read case: N
case in core, id=0001, studycode RRUN, interviewer ID bkok : deleted-N,
error-N, altered-N, written-N, new-N, changed-0
TRfile name c:\acme\road\rrunr.tr (opt=111)
Number of cases: 500, adds: 500, dels: 0, updt: 4
The case length this file has is 640
The comment this file has is '<blank>'
This file's style is new w/o directory
no OUTPUTfile
There are 1 of a maximum of 10 DB files opened
Listed below are the DB files open
They are listed in search order.
c:\acme\road\rrunr.db
filesCHAIN dump from 005a14fc
at 005a14fc: #1=5: f:\cfmc70\control\msg9607 (opt=1) ... buf: 512
at 005a1e10: #2=-1: TE205000 (opt=33) ... buf: 20000
at 005a1118: #3=999: (console) (opt=20081) ... buf: 0
at 005a1b84: #4=999: (console) (opt=100082) ... buf: 0
at 005a1054: #6=7: cfmcsave (opt=100082) ... buf: 2048
at 005a81d3: #7=6: c:\acme\road\rrunr.db (opt=401) ... buf: 2048
at 005a8eb0: #8=8: c:\acme\road\tori.prt (opt=100082) ... buf: 2048
at 005a9214: #10=9: c:\acme\road\rrunr.tr (opt=111) ... buf: 2048
farmem 1K blocks: 275 max blocks: 1 (275000 bytes)
>printfile chain
file c:\acme\road\red.prt: #0=red.
```

>S_TIME

Sets a timing function needed by CfMC programs. This time will already be set by

the program on most machines, but 486 machines or machines with plasma screens (like COMPAQ portable or many laptops) will need to have >S_TIME set manually.

Syntax:

```
>STIME ######
```

Determine the current S_TIME by running a CfMC program and then typing >S_TIME and pressing Enter. This is not an exact value, and it can vary a little each time you check it. It will be about the same on systems with the same processor (286s, 386s, etc.). To set the S_TIME, enter the time as a six-digit number as HHMMSS. You can also specify the S_TIME in the INITIAL file in CFMC\CONTROL (DOS, UNIX; or CONTROL.CFMC in MPE).

>STOP_WATCH

Displays timing information in minutes and seconds. This command works in tandem with the ~STOP_WATCH command. Two numbers are returned; the first is the elapsed time since the beginning of the run, and the second is the elapsed time since the last >STOP_WATCH or ~STOP_WATCH command. You will get a total timing at the end of the run regardless of the use of >STOP_WATCH or ~STOP_WATCH.

Syntax:

```
>SW
```

Example:

```
>SW
total time: 0 minutes 6 seconds
>SW
total time: 0 minutes 11 seconds
(and for part 2, 0 minutes 5 seconds)
```

>STUDY_NAME

Establishes the study name. This name replaces the asterisk in ~INPUT *, >USE_DB *, >PRINT_FILE *, and [*] (to indicate the study name on the header statement in PREPARE).

Syntax:

>STUDY name

Option:

name One to six alphanumeric characters, where the first character must be alpha.

Example:

```
>STUDY BANK
>USEDB *
```

The program will use the DB file BANK.

>SYSTEM <command>

Allows you to specify system commands (such as DIR) in your spec file. A meta command specified without >SYSTEM will be checked first as a CfMC meta command. The program prints a warning if the command violates CfMC syntax before passing it to the operating system. This is the default. Suppress the warning by saying >-SYSTEM.

Syntax:

```
>SYS <meta command>
```

Option:

<meta command> Distinguish between a CfMC meta command and an operating system command of the same name (like ECHO) by saying >ECHO or >SYS ECHO respectively. Or, you can use >SYS !, which
prevents any non-CfMC meta commands from being passed to the operating system.

Example:

```
>SYS DIR *.TR
```

You can put this command in the CfMC INITIAL file (\CFMC\CONTROL\INITIAL in DOS, UNIX; INITIAL.CONTROL.CFMC in MPE) so that it will be in effect whenever a CfMC program is loaded.

>TAB_WARN

Controls whether the warning “converting tabs to spaces” for text lines in spec files prints or not. Default is on, so use >-TABWARN to suppress the warning. Mentor converts tabs to spaces (one space per tab).

Syntax:

```
>-TABWARN
```

See also: >CHARACTERSET=

>TRANSLATE

Translates one ASCII character to another.

Syntax:

```
>TRANS ASCII_value ASCII_value
```

Example:

```
>TRANS 65 97
```

This will change all capital "A"s (ASCII 65) to lower case "a"s (ASCII 97). A separate translate command must appear for each different character to be translated. The >TRANSLATE command operates on all characters which appear in the spec following it, not just those that are a part of titles, stubs, etc.

Note: *Be careful not to issue a >TRANSLATE command that will change Mentor commands and keywords. This command not only affects text you enter at the keyboard, but also affects text generated by Mentor, such as automatic labels.*

Tables may be run and stored in a DB file, then a >TRANSLATE command issued just before the load and print phase of a run, as in the example below.

Example:

```
>USE_DB trans1
>TRANSLATE 65 97
>TRANSLATE 66 98
~EXECUTE load=T001 print
~END
```

If you want to translate a character that has an ASCII value greater than 126 [for example, American dollar signs (ASCII 36) to British pound signs (ASCII 156)], you must include the >ALLOW_ALL_CHARACTERS meta command in your spec file.

>USE_DB

Opens a DB file. You can have multiple DB files open with write access and can put items directly into any of the 'save to' DB files (see example under >CREATE_DB).

Syntax: `>USEDB dbname,roption,EC,D=doption`

Options:

dbname The only required parameter. You can specify " * " as the dbname in conjunction with a name specified on a >STUDY_NAME command.

roption Read option, either RO or RW. READ_ONLY(RO), lets you read entries from the DB file. This is the default. READ_WRITE(RW) lets you read entries from the DB file and write (and save) entries to the DB file.

EC ECHO displays a message when you get an item from the DB file. The default is to not echo.

DUPLICATE= Determines what the program will do when it encounters duplicate names.

doptions:

ERROR prevents you from storing a duplicate item and displays an error message (this is the default).

QUIET replaces the existing entry with the new one of the same name, without displaying a warning.

REPLACE replaces the existing entry with a warning (like D=W) but helps keep your DB file from getting filled up with multiple versions of items since it deletes the old entry and, if the new entry is the same size or smaller, uses that same space again.

WARN replaces the existing entry with the new one of the same name, but displays a warning message.

>USE_DB

Opens a DB file. You can have multiple DB files open with write access and can put items directly into any of the 'save to' DB files (see example under >CREATE_DB).

>Var(iable)_len(gth)_ascii

This controls whether or not ASCII files on the HP3000 are built with a variable length or fixed length.
>-Var_len_ascii (with the minus) will make them a fixed length.

Appendix B
Allowed Abbreviations

COMMANDS AND ABBREVIATIONS

This is an alphabetical list of all CfMC commands and their abbreviations. There may be other abbreviations of the commands that the program will accept, but these are the abbreviations that have been tested. The reference number will tell you what type of command it is and which manual to look it up in: Survent (ref. no. 1-10), Mentor (ref. no. 11-22) or Utilities (ref. no. 31). Here are the basic rules for abbreviations:

- Underscores used to separate words are always optional. For example: CHECK_EXIST can be abbreviated CHECKEXIST.
- Plural commands can be abbreviated without the S. For example, COLORS can be abbreviated COLOR.
- Functions can be abbreviated, but must always include the open parenthesis. For example, STRING_LENGTH (is abbreviated STRLEN).
- Standard abbreviations are:

CLEAN	CLN
COLUMN	COL
ERROR	ERR
LENGTH	LEN
NUMBER	NUM
PRINT	PRT
TOTAL	TOT
WIDTH	WID
WEIGHT	WT

The following table provides Survent and Mentor references:

Survent		Mentor	
1	Study Header option	11	Tilde commands
2	Interview Control (3.2.1)	12	CLEANER keywords allowed in a procedure
3	Compose Control (3.2.2)	13	CLEANER keywords not used in procedure
4	Compile Control (3.2.3)	14	DEFINE main keywords
5	Data Control (3.2.4)	15	Print #information# in printed text
6	Data Entry Question Types (2.4)	16	Procedure controls
7	Control Statements (3.1)	17	Variable types
8	Composing and compiling questions in PREPARE (2.3.1)	18	Constants
9	The PHONE Statement (6.1.1)	19	Objects
		20	Functions
		21	Region keywords
		22	Spec generation control (4.7)
		31	Meta

LIST OF ABBREVIATIONS

An alphabetical list of abbreviations is available on CfMC's BBS.

Item	Abbreviation	Reference
Absolute_value	ABS(	20
Adjust	ADJ	11
Input_text_location	INTEXTLOC	
New_length	NEWLEN	
Number_of_cases	NUMC	
Output_text_location	OUTTEXTLOC	
All	ALL	21
Allow_abort	ABORT	1,2
Allow_all_character	ALLOWALLCHAR	31
Allow_backup	ALLOWBACKUP	2
Allow_indent	ALLOWIND	31
Allow_monitor	ALLMON	2
Allow_reset	ALLOWRESET	2
Allow_retake	ALLOWRETAKE	2
Allow_suspend	ALLOWSUSP	2
Allow_terminate	ALLOWTERM	2
Allow_text_edit	ALLOWTEXEDT	2
Alter_flag	ALTFLAG	18
Answer_length=	ANSLEN=	1
Automatic_increment	INC	1
Auto_punches	AUTOP	3
Auto_response_code	AUTORESP.	3
Auto_return	AUTORET	2
Average(	AVG(	20
Backup_here	BACKUPH	2
Backup_where_at	BACKUPW	2
Balance(	BAL(	20
Batch_job	BATCH	31
Beep	BEEP	31
Blank_lines=	BLANKLINE=	3

Item	Abbreviation	Reference
Block	BLOCK	5
Boolean	B	17
Boss_dot=	BOSSDOT=	31
Box	BOX	31
Browse	BROWSE	31
By	BY	21
Cache_message..	CACHEMSG	31
Call_dos	DOS	31
Card_format=	CARD=	1
Cascade(	CASCADE(	20
Case_compare(	CASECOMP(	20
Case_id.	CASEID	18
Case_id=	ID=	1
Case_length	LEN	1
Case_number	CASENUMBER	18
Case_written	CASEWRITTEN	18
Categories	C	17
Categories(	CATS(	18
Category	CAT	6
Category_function(	CFUNC(	20
Cfmc_file_extension	CFMCEXT	31
Check_column_overlap	CHKOVERLAP	4
Check_exist	CHECKEXIST	31
Check_file	CHK	1
Cleaner	CLN	11
Alter	ALT	12
Assign_delete_flag	DELETE	12
Assign_error_flag	ASSIGNERR	12
Blank	B	12
Check	CHK	12
Check_columns	CHKCOL	12
Choose_file	CHOOSE	12
Clean	CLN	12
Clear_error_flag	CLRERR	12

Item	Abbreviation	Reference
Copy	COPY	12
Create_table	CREATE	12
Define	DEF	13
Display_ascii	D	12
Display_binary	DB	12
Display_column	DC	12
Display_text	DT	12
Do_meta	META	12
Do_set	SET	12
Do_tables	TAB	12
Drop	DROP	13
Dump_variables	DUMP	12
Edit	EDIT	12
End_when	ENDWHEN	12
Enter	ENTER	12
Erase_tex	ET	12
Error	ERR	12
Execute	EXC	12
Execute_any	EXANY	12
Execute_data	EXDATA	12
Execute_eof	EXEOF	12
File	FILE	13
Fill	FILL	12
Find	F	13
Find_flagged	FF	13
Find_flagged_redo	FFR	13
Find_redo	FR	13
Fix_up	FIX	12
Goto	GOTO	12
Halt	HALT	12
Hun	H	13
Interview	INT	12
Load_tables	LOAD	12
Make_data	KDATA	12
Modify	M	12

Item	Abbreviation	Reference
Modify_ascii	MA	12
Modify_binary	MB	12
Modify_column	MC	12
Modify_text	MT	12
New_case	NEW	12
Next	N	12
Next_redo	NR	13
No_update	NOUPDATE	12
Null	NULL	12
Ok_columns	OKCOL	12
Pause	PAUSE	12
Print_lines	PRT	12
Print_tables	PRTTAB	12
Print_to_data	PRTDATA	12
Put_id	PUTID	12
Redo	R	13
Restore	RESTORE	13
Say	S	12
Set	SET	13
Show	SHOW	12
Show_tables	SHOWTAB	12
Skip_to	SKIP	12
Store_table	STORE	12
Terminal_print	TERMPR	12
Terminal_say	TERMSAY	12
Terminate	TERM	12
Transfer	T	12
Undelete	UNDELETE	12
Unload_tables	UNLOAD	12
Update	UPDATE	13
Upshift	UP	12
View	VIEW	13
When	WHEN	12
Write_case	WRITE	12
Write_qsp	WRITEQSP	12

Item	Abbreviation	Reference
Yes_update	YESUPDATE	12
Zspc	ZSPC	12
Clear_screen	CLS	31
Close_db	CLOSEDB	31
Close_qff	CLOSEQFF	31
Colors	COLOR	31
Column	COL	5
Column_kick	COLKICK	5
Comment	COM	11
Complete(	COMPLETE(	20
Control_y_quiet	CTRLYQ	31
Convert(	CVT(	19
Copy	COPY	11
Keep_deletes	KPDEL	
Copy	COPY	31
Dashboard	DASHBOARD	1
Data_compare(	DATACOMP(	20
Data_location_required	LOCREQ	1
Date_time	DATETIME	18
Db_file	DB	1
Db_sizes=	DBSIZE=	31
Db_status	DBSTATUS	31
Db_to_file	DBTOFILE	31
Define	DEF	31
Define	DEF	11
Axis=	AXIS=	14
Base	BASE	
Break	BRK	
Break_control=	BRKCTRL=	
Do_statistics	STAT	
Effective_n	EFFN	
Frequency	FREQ	
Maximum	MAX	
Mean	MEAN	

Item	Abbreviation	Reference
Mean_frequency	MFREQ	
Median	MED	
Minimum	MIN	
Net_overlay	NOLAY	
Null_break	NULLBRKKCTRL=Do_	
Overlay	OLAY	
Percentile=	PTILE=	
Raw_count	RAWCOUNT	
Se	SE	
Std	STD	
Sum	SUM	
Variance	VAR	
Weight	WT	
Banner=	BAN=	14
Edit=	EDIT=	14
All_possible_pairs_test	ALLPAIRTEST	
Anova	ANOVA	
Anova_scan	ANOVASCAN	
Bottom_margin=	BOTM=	
Call_table=	CALLTAB=	
Check_cells	CHKCELL	
Chi_square	CHISQ	
Chi_square_anova_format	CHISQANOVAFORM	
Clear_sub_totals	CLRSUBTOT	
Column_info=	COLINFO=	
Blank	B	
Column=	C=	
Column_statistics_values	COLSTAT	
Cumulative_percent	CP	
Frequency	FREQ	

Item		Abbreviation	Reference
	Frequency _decimals=	FD=	
	Horizontal _percent=	HP=	
	Leave_table_open	LVTABOPEN	
	Minimum_base=	MINB=	
	Number_format=	NUMFORM=	
	Percent_decimals=	PD=	
	Percent_sign	PS	
	Put_characters=	PUT=	
	Statistics_column	S	
	Statistics_decimals=	SD=	
	Vertical_percent	VP	
	Width=	W=	
Column_mean		MEAN	
Column_median=		MED=	
Column_na		COLNA	
Column_se		SE	
Column_sigma		SIG	
Column_statistics_values=		COLSTAT=	
Column_std		STD	
Column_tna		COLTNA	
Column_variance		VAR	
Column_width=		CWID=	
Continued_Location=		CONT=	
	Bottom_centered	BOTC	
	Bottom_left	BOT	
	Bottom_right	BOTR	
	None	NONE	
	Top_centered	TOPC	
	Top_left	TOP	
	Top_right	TOPR	
Continued_Number=		CONTNUM=	
	After_continue	AFTERCONTINUE	

Item		Abbreviation	Reference
	After_table_name	AFTERTABLENAME	
	None	NONE	
Cumulative_percent		CPER	
Data_indent=		DATAIND=	
Do_printer_statistics		PRTSTAT	
Do_statistics		STAT	
Do_statistics_tests=		STATTEST=	
Empty_cells=		EMPTYCELL=	
Extra_rows_ok		XROWOK	
Extra_stubs_ok		XSTUBOK	
Fisher		FISHER	
Flag_minimum_base_cell		FLAGMINBASE	
Frequency		FREQ	
Frequency_decimals=		FDEC=	
Frequency_only.		FREQONLY	
Horizontal_percent		HPER	
Leave_enhancements_on		LEAVEENHANCEMENTSON	
Leave_page_open		LVPGOPEN	
Leave_table_open		LVTABOPEN	
Mark_chi_square		MARKCHISQUARE	
Minimum_base=		MINBASE=	
Minimum_for_printing=		MINFORPRT=	
Minimum_frequency=		MINFREQ=	
Minimum_indent_left		MININDLEFT	
Minimum_percent=		MINPER=	
Newman_keuls_test		NKTEST	
Number_format=		NUMFORM=	
Number_of_cases		NUMC	
Overline=		OLINE=	
Page_length=		PGLEN=	
Page_number		PGNUM	
Page_width=		PGWID=	
Paired_variance		PAIREDVAR	
Percent_decimals=		PDEC=	
Percent_sign		PERSIGN	

Item	Abbreviation	Reference
Pooled_variance	POOLEDVAR	
Prefix=	PREF=	
Print_alpha_table_names	PRTATABNAME	
Print_blank_percent_lines	PRTBLKPER	
Put_characters=	PUTCHAR=	
Rank_column_base=	RANKCOL=	
Rank_if_indicated	RANKIFIND	
Rank_level=	RANK=	
Rank_order=	RANKORDER=	
Ascending	A	
Descending	D	
Row_mean	ROWMEAN	
Row_median=	ROWMED=	
Row_na	ROWNA	
Row_se	ROWSE	
Row_statistics_values=	ROWSTAT=	
Row_std	ROWSTD	
Row_tna	ROWTNA	
Row_variance	ROWVAR	
Running_lines=	RUNLINE=	
Save_rank_info=	SAVERANK=	
Separate_variance	SEPARATEVAR	
Show_significance_only	SHOWSIGONLY	
Significance_level	SIGLEV	
Skip_lines=	SKIP=	
Star_percent=	STARPER=	
Stats_footnote	STATSFOOTNOTE	
Statistics_decimals=	SDEC=	
Statistics_percent_error	STATPERERR	
Statistics_percent_warn	STATPERWARN	
Status	STATUS	
Stub_default=	STUBDEF=	
Stub_extra=	STUBEX=	
Stub_indent=	STBIND=	
Stub_preface=	STUBPREF=	

Item		Abbreviation	Reference
	Stub_rank_indent=	STUBRIND=	
	Stub_suffix	STUBSUF=	
	Stub_width=	SWID=	
	Stub_wrap_indent=	STUBWIND=\	
	Word	WORD	
	Subtotal_1	SUBTOT	
	Subtotal_2	SUBTOT2	
	Subtotal_3	SUBTOT3	
	Subtotal_4	SUBTOT4	
	Subtotal_5	SUBTOT5	
	Suffix=	SUF=	
	Suppress_rows_base=	SUPBASE=	
	Table_tests=	TABTEST=	
	Tcon=	TCON=	
	Footer	FOOT	
	Header	HEAD	
	Indent=	INDENT=	
	Print_page_numbers	PRTGNUM	
	Table_names	TABNAME	
	Tcon_page_numbers	TCONPGNUM	
	Title	TITLE	
	Title_2	T2	
	Title_4	T4	
	Title_5	T5	
	Tfrp	TFRP	
	Title_4_for_base	T4BASE	
	Top_margin=	TOPM=	
	Underline=	ULINE=	
	Use_rank_info=	USERANK=	
	Vertical_percent	VPER	
Edit_variable=		EDITVAR=	14
Footer=		FOOT=	14
Header=		HEAD=	14
Lines=		LINE=	14

Item	Abbreviation	Reference
Date	DATE	15
Page_number	PGNUM	15
Table_name	TAB	15
Time	TIME	15
Variable=	VAR=	15
Prepare=	PREP=	14
Procedure=	PROC=	14
Else	ELSE	16
End_if	ENDIF	16
End_while	ENDWHILE	16
If	IF	16
While	WHILE	16
Statistics=	STAT=	14
Stub=	STUB=	14
Base_row	BASE	
Column_statistics_values	COLSTAT	
Comment	COM	
Cumulative_percent	CPER	
Data_indent=	DATAIND=	
Density	DENSITY	
Do_sig_t	SIGT	
Print_mean	PRTMEAN	
Do_statistics=	STAT=	
All_possible_pairs	ALLPAIRTEST	
Anova_scan	ANOVASCAN	
Fisher	FISHER	
Kruskall _wallis_test	KWTEST	
Newman _keuls_test	NKTEST	
Repeated_measures	RM	
Do_t_test	TTEST	
Print_mean	PRTMEAN	
For_out_table	FOROUTTABLE	
Frequency	FREQ	

Item	Abbreviation	Reference
Frequency_decimals=	FDEC=	
Frequency_only	FREQONLY	
Horizontal_percent	HPER	
Keep_rank_control=	KR=	
Keep_subtotal_1	KPSUBTOT	
Keep_subtotal_2	KPSUBTOT2	
Keep_subtotal_3	KPSUBTOT3	
Keep_subtotal_4	KPSUBTOT4	
Keep_subtotal_5	KPSUBTOT5	
Lines_left=	LINELEFT=	
Long_comment	LCOM	
Minimum_frequency=	MINFREQ=	
Minimum_percent=	MINPER=	
New_page	NEWPG	
Number_format=	NUMFORM=	
Overline	OLINE	
Percent_decimals=	PDEC=	
Percent_sign	PERSIGN	
Print_row=	PRT=	
Any_response	AR	
Chi_square	CHI	
Mean	M	
Median	MED	
No_answer	NA	
Se	SE	
Sigma	SIG	
Std	STD	
Subtotal_1_clear	SUBTOT	
Subtotal_2_clear	SUBTOT2	
Subtotal_3_clear	SUBTOT3	
Subtotal_4_clear	SUBTOT4	
Subtotal_5_clear	SUBTOT5	
Subtotal_1 _no_clear	SUBTOTNC	
Subtotal_2		

Item	Abbreviation	Reference
_no_clear	SUBTOT2NC	
Subtotal_3		
_no_clear	SUBTOT3NC	
Subtotal_4		
_no_clear	SUBTOT4NC	
Subtotal_5		
_no_clear	SUBTOT5NC	
Super_sigma	SSIG	
Suppressed	SUP	
Total	TOT	
Unweighted_any		
_response	UAR	
Unweighted_no		
_answer	UNA	
Unweighted_total	UTOT	
Variance	V	
Put_characters=	PUTCHAR=	
Rank_level=	R=	
Rank_pre_group	RPREG	
Reprint_base	PRTBASE	
Sigma	SIGMA	
Skip_lines=	SKIP=	
Statistics_decimals=	SDEC=	
Statistics_row	STAT	
Stub_extra	STUBX	
Stub_indent=	STBIND=	
Subtotal_1	SUBTOT	
Subtotal_2	SUBTOT2	
Subtotal_3	SUBTOT3	
Subtotal_4	SUBTOT4	
Subtotal_5	SUBTOT5	
Suppress	SUP	
Suppress_if_edit_frequency	SUPFREQ	
Suppress_if_edit_ no_frequency	SUPNOFREQ	

Item		Abbreviation	Reference
	Suppress_rows_base=	SUPBASE=	
	Underline	ULINE	
	Vertical_percent	VPER	
Table=		TAB=	14
	Table_set=	TABSET=	14
	Table_specs=	TABSPEC=	14
	Title=	TITLE=	14
	Variable=	VAR=	14
Delete		DEL	31
Delete_db_item		DELDBITEM	31
Delete_flag		DELFLAG	18
Demo		DEMO	31
Disk_room		DISKROOM	31
Display		DISP	7, 22
Do_all		DOALL	31
Do_mentor		DOMentor	4
Do_mentor		Mentor	22
Do_variables		VARS	4
Drop		DROP	21
Dud		DUD	18
Dump		DUMP	31
Dump_level		DUMPLEVEL	31
Duplicate_labels		DUPLAB	1
Echo		ECHO	1, 31
Echo_cats		ECHOCATS	2
Echo_defines		ECHODEF	31
Edit (header option)		ED	1
Edit (question type)		EDI	7
Edit (question type)		EDT	7
Edit		EDIT	3, 31
Edit_errors		EDITERR	3
Edit_file		EDITF	31
Edit_previous		EP	31
Else		ELSE	31

Item	Abbreviation	Reference
Else_if	ELSEIF	31
End_block	ENDBLOCK	5
End_grid	ENDGRID	2
End_if	ENDIF	31
End_loop	ENDLOOP	7
End_of_file	EOF	31
End_remove	ENDREMOVE	4
End_repeat	ENDREP	31
End_resume	ENDRES	2
End_rotate	ENDROT	2
End_special	ENDSPECIAL	2
End_suspend	ENDSUSP	2
Eof_data.	EOFDATA	18
Error_beep	BEEP	1
Error_flag	ERRFLAG	18
Errors	ERRORS	18
Error_line_number	ERRLNNUM	31
Error_stop	ERRSTOP	4
Execute=	EXC	11
Banner=	BAN=	
Base=	BASE=	
Build_tables	BUILDTAB	
Close_page	CLOSEPG	
Column=	COL=	
Column_short_weight=	COLSHORTWT=	
Column_weight=	COLWT=	
Drop_chain	DROPCH	
Edit=	EDIT=	
Filter=	FIL=	
Footer=	FOOT=	
Global_edit=	GLOBALEDIT=	
Header=	HEAD=	
Load_table=	LOAD=	
Local_edit=	LOCEDIT=	
Make_tables	MKTAB	

Print_all	PRTALL	
Print_run	PRTRUN	
Print_table	PRT	
Procedure=	PROC=	
Read_procedure=	RDPROC=	
Reset	RESET	
Retab=	RETAB=	
Row=	ROW=	
Row_short_weight=	ROWSHORTWT=	
Row_weight=	ROWWT=	
Run_chain	RUNCH	
Set	SET	
Show_chain	SHOWCH	
Statistics=	STAT=	
Status	STATUS	
Store_table=	STORE=	
Stub=	STUB=	
Stub_preface=	STUBPREF=	
Stub_suffix=	STUBSUF=	
Table=	TAB=	
Table_set=	TABSET=	
Table_specs=	TABSPEC=	
Title=	TITLE=	
Title_2=	T2=	
Title_4=	T4=	
Title_5=	T5=	
Weight=	WT=	
Exclusive	XCLUSIVE	22
Exponent(	EXP(	20
Expression	EXP	7
Fake_time	FAKETIME	31
False	FALSE	18
Field	FLD	6
File_compare(	FILECOMP(	20
File_to_db	FILETODB	31

Item	Abbreviation	Reference
Fill_defines_inside_quotes	FILLDEF	31
First_case	FIRSTCASE	18
First_value(	FIRSTVAL(	20
Fix	FIX	2
Flag_disallowed_cats	FLAG	1
Flip(	FLIP(	20
Fone_text_length=	FONETEXT=	1
Freak	FREAK	11
Frequency	FREQ	11
Fsig(	FSIG(	20
Function(	FUNC(	20
Generate	GEN	7
Goto	GOT	7
Goto (tilde)	GOTO	11
Grid	GRID	2
Group	GRP	2
Halt	HALT	31
Any	ANY	
Error	ERR	
None	NONE	
Warning	WARN	
Hard_code	HARDCODE	4
Hard_copy	HARDCOPY	4
All	ALL	
Date	DATE	
Lines=	LINE=	
Page	PG	
Page_length=	PGLEN=	
Page_title= Top/Bottom	PGT=TOP/BOT	
Printer=	PRT=	
Show_cat_as	SCATA	
Show_cat_ns	SCATN	
Show_labels	SLAB	
Show_logic	SLOGIC	
Show_qq_nums	SQQNUM	

Item	Abbreviation	Reference
Show_sameas	SSAMEAS	
Hardcopy_file	HRD	1
Help	HELP	31
Help_cat	HELPCAT	10
Help_display	HELPDISP	10
Help_echo_m	HELPECHOM	10
Help_echo_single	HELPECHOS	10
Help_edit	HELPEDIT	10
Help_fld	HELPFLD	10
Help_grid	HELPGRID	10
Help_hilite_multi	HELPHILITEM	10
Help_hilite_single	HELPHILITES	10
Help_num	HELPNUM	10
Help_reset	HELPSET	10
Help_text	HELPTEXT	10
Help_var	HELPVAR	10
Help_var_l	HELPVARL	10
Help_var_n	HELPVARN	10
Hide_all.	HIDEALL	4
Highlight_cats	HILT	2
High_point	HIPT	5
If_defined	IFDEF	31
Include	INCLUDE	8
Info_between	INFOBETWEEN	1
Input	IN	11
Allow_new.	NEW	
Allow_update	UPDATE	
Ascii=	ASCII=	
Backup	BUP	
Binary=	BIN=	
Buffer_size	BUFSIZE	
Case_location	CASELOC	
Comment	COM	
Create	CR	

Item	Abbreviation	Reference
Dots	DOT	
Drop_blank_line	DROPBLK	
Drop_changes	DROPCHANGE	
Dta_format	DTA	
Exclusive	XCLUSIVE	
Ignore_directory	IGNOREDIR	
Join=(	JOIN=(	
Maybe_backup	MAYBEBUP	
Maybe_create	MAYBECR	
New_buffer	NEWBUF	
Number_input_buffers	NUMBUF	
Number_of_cases	NUMC	
Protect=	PROTECT=	
Quit_on_blank_lines	QUITONBLK	
Read_control=	RDCTRL=	
Read_first_case	READCASE	
Records_per_case	RECSPERCASE	
Salvage_data	SALVAGE	
Select	SELECT	
Server_acces	SERVER	
Share	SHARE	
Short_line_warn	SHORTWARN	
Stop_after	STOP	
Study_name=	STUDY=	
Swapped_binary=	SWAPBIN=	
Text_location	TEXTLOC	
Total_length	TOTLEN	
Uncompressed	UNPRESS	
Use_deleted	USEDEL	
Verify	VERIFY	
Work_length	LEN	
Write_share	WRITESHARE	
Interview	INT	11, 21
Join_columns	JOINCOL(	20

Item	Abbreviation	Reference
Join_rows(	JOINROW(	20
Julian_date	JULIANDATE	18
Key_delay	KEYDELAY	31
Lang=	LANG=	1
Last	LAST	21
Last_case	LASTCASE	18
Last_value(	LASTVAL(	20
Line_number	LNNUM	18
List_db_contents	LISTDB	31
Append	APPEND	
Pattern=	PAT=	
Sort=	SORT=	
Template=	TEMP=	
Type=	TYPE=	
All	ALL	
Tables	TAB	
Variables	VAR	
Loaded(	LOAD(	20
Location_format	LOCFORM	31
Logarithm(	LOG(	20
Logging	LOG	1
Loop	LOO	7
Make_asq	MKASQ	21
Make_boolean(	MKB(	19
Make_categories(	MKC(	19
Make_number(	MKN(	19
Make_read_control	MKRDCTRL.	11
Make_specfiles	SPEC	4
Make_string(	MKS(	19
Make_vector(	MKV(	19
Math_values	MATHVALUES	18
Maximum_labels=	MAXLAB =	1
Maximum_quota_number=	MAXQ=	1

Item	Abbreviation	Reference
Maximum_qfile_size	MAXQFILESIZE	1
Maximum_question_size	MAXQSTSIZE	1
Maximum_subscript(	MAXSUB(	20
Maximum_value(	MAX(	20
Mentor_cln_file	MentorCLN	22
Mentor_def_file	MentorDEF	22
Mentor_demo	MDDEMO	31
Mentor_spec	Mentor	22
Mentor_tab_file	MentorTAB	22
Minimum_subscript(	MINSUB(	20
Minimum_value(	MIN(	20
Missing	MISSING	18
Modify_caseid	MODID	1
Modify_fone_file	MODFONE	1
Modify_quota_file	MODQUO	1
Monitor	MONITOR	1
Na	NA	21
Net(	NET(	20
New	NEW	11
Next	NEXT	11
Next_case_id=	NEXTID=	1
Not	NOT	19
Number	N	17
Number_adjustment	NUMADJ	31
Number_function(	NFUNC(	20
Numbering	NUMBER	4
Number_of_cases=	NUMCASES=	1
Number_of_columns(	NUMCOL(	20
Number_of_items(	NUMITEM(	20
Number_of_rows(	NUMROW(	20
Numbers_from_table(	NUMFRTAB(	20
Numeric	NUM	6
Numeric_width_required	NUMWR	1
One_interview	ONEINT	1

Item	Abbreviation	Reference
Output	OUT	11
Ascii	ASCII	
Binary	BIN	
Cards_image	CARD	
Case_length	LEN	
Comment	COM	
Dta_format	DTA	
Hex.	HEX	
Maybe_create	MAYBECR	
Number_of_cases	NUMC	
Omit_directory	OMITDIR	
Study_name=	STUDY=	
Swapped_binary	SWAPBIN	
Swapped_hex	SWAPHEX	
Trim_blanks	TRIMBLANK	
Uncompressed	UNPRESS	
Write_now	WRITENOW.....	
Page_number	PGNUM	18
Password=	PASS=	1
Phone	PHO	9
Practice	PRACTICE	11
Prepare	PREP	8
Compile	COMPILE	
Cleaning_specs	CLN	
C_mentor_specs	CMentor	
C_survent_specs	CSPEC	
Persee_specs	PERSEE	
Quantum_specs	QUANTUM	
Specs	SPEC	
Spl_mentor_specs	SPLM	
Spl_survent_specs	SPLS	
Spss_specS	SPSS	
Disk_based_response_list	DBR	
Make_spec_files	MKFILE	
Check_file	CHK	

Item	Abbreviation	Reference
Hardcopy_file	HRD	
Qsp_file	QSP	
Sum_file	SUM	
Print_file	PRTF	31
Append	APPEND	
Bottom_margin=	BOT=	
Copies=	COPIES=	
Echo	ECHO	
File_name=	FILENAME=	
Form_feed	FF	
Forms=	FORMS=	
Header_page	HEADPG	
Laser_control	LASERCTRL	
Bold_off	BOLDOFF	
Bold_on	BOLDON	
Close_string=	CLOSESTR=	
Color_background_black=	CBBLK=	
Color_background_blue=	CBBLU=	
Color_background_cyan=	CBCYAN=	
Color_background_default=	CBDEF=I	
Color_background_green=	CBGRN=	
Color_background_magenta=	CBMAGENTA=	
Color_background_red=	CBRED=	
Color_background_white=	CBWHT=	
Color_background_yellow=	CBYEL=	
Color_foreground_black=	CFBLK=	
Color_foreground_blue=	CFBLU=	
Color_foreground_cyan=	CFCYAN=	
Color_foreground_default=	CFDEF=	
Color_foreground_green=	CFGRN=	
Color_foreground_magenta=	CFMAGENTA=	
Color_foreground_red=	CFRED=	

Item		Abbreviation	Reference
	Color_foreground_white=	CFWHT=	
	Color_foreground_yellow=	CFYEL=	
	Control_size=	CTRLSIZE=	
	Escape_character=	ESCCHAR=	
	Extra_width=	XTRAWID=	
	Flashing_off=	FLASHOFF=	
	Flashing_on=	FLASHON=	
	Initial_file=	INITFILE=	
	Initial_size=	INITSIZE=	
	Initial_string=	INITSTR=	
	Inverse_off=	INVERSEOFF=	
	Inverse_on=	INVERSEON=	
	New_line=	NEWLN=	
	New_page=	NEWPG=	
	Page_length=	PGLEN=	
	Page_width=	PGWID=	
	Underline_off=	ULINEOFF=	
	Underline_on=	ULINEON=	
	User_a_off=	USERAOFF=	
	User_a_on=	USERAON=	
	User_b_off=	USERBOFF=	
	User_b_on=	USERBON=	
	User_c_off=	USERCOFF=	
	User_c_on=	USERCON=	
	User_d_off=	USERDOFF=	
	User_d_on=	USERDON=	
	User_e_off=	USEREOFF=	
	User_e_on=	USEREON=	
	User_f_off=	USERFOFF=	
	User_f_on=	USERFON=	
	User_g_off=	USERGOFF=	
	User_g_on=	USERGON=	
	User_h_off=	USERHOFF=	
	User_h_on=	USERHON=	
	User_i_off=	USERIOFF=	

Item		Abbreviation	Reference
	User_i_on=	USERION=	
	User_j_off=	USERJOFF=	
	User_j_on=	USERJON=	
	User_k_off=	USERKOFF=	
	User_k_on=	USERKON=	
	User_l_off=	USERLOFF=	
	User_l_on=	USERLON=	
	User_m_off=	USERMOFF=	
	User_m_on=	USERMON=	
	User_n_off=	USERNOFF=	
	User_n_on=	USERNON=	
	User_o_off=	USEROOFF=	
	User_o_on=	USEROON=	
	User_p_off=	USERPOFF=	
	User_p_on=	USERPON=	
	User_q_off=	USERQOFF=	
	User_q_on=	USERQON=	
	User_r_off=	USERROFF=	
	User_r_on=	USERRON=	
	User_s_off=	USERSOFF=	
	User_s_on=	USERSON=	
	User_t_off=	USERTOFF=	
	User_t_on=	USERTON=	
	User_u_off=	USERUOFF=	
	User_u_on=	USERUON=	
	User_v_off=	USERVOFF=	
	User_v_on=	USERVON=	
	User_w_off=	USERWOFF=	
	User_w_on=	USERWON=	
	User_x_off=	USERXOFF=	
	User_x_on=	USERXON=	
	User_y_off=	USERYOFF=	
	User_y_on=	USERYON=	
	User_z_off=	USERZOFF=	
	User_z_on=	USERZON=	

Item	Abbreviation	Reference
Wide_off=	WIDEOFF=	
Wide_on=	WIDEON=	
Laser_number	LASERNUM	
Page_length=	PGLEN=	
Page_width=	PGWID=	
Top_margin=	TOP=	
User	USER	
Print_file_defaults	PRTFDEF	31
Print_repeat	PRTREP	31
Prn	PRN	31
Purge	PURGE	31
Purge_same	PURGESAME	31
Put_characters=	PUTCHAR=	31
Qff_file	QFF	1, 11
Qff_file_name=	QFF=	1
Qsp_file	QSP	1
Queue=	QUEUE=	31
Qui	QUIT	31
Quota	QUO	7
Quota_file	QUO	1
Random_category(	RANDOMCAT(	20
Random_seed=	RANDOMSEED=	31
Random_sequence(	RANDOMSEQ(	20
Random_value	RANDOMVAL	18
Rank_table_columns	RANKTABCOL	20
Read	READ	31
Read_named_quotas	READNAMEDQUOTAS	2
Reformat	RFT	11
Append_cat_data	APPEND	
Card_image	CARD	
Check_file	CHK	
Cleaning_specs	CLN	
C_mentor_specs	CMentor	

Item	Abbreviation	Reference
Core_block_size=	CORE=	
C_survent_specs	CSPEC	
Do_non_text_data	DONTEX	
Do_text_data	DOTEX	
Expand_single_cats	EXPAND	
Hardcopy_file	HRD	
Lotus_format	LOTUS	
Map_file	MAP	
Persee_specs	PERSEE	
Qsp_file	QSP	
Quantum_specs	QUANTUM	
Single_cat_response	RESPONSE	
Specs	SPEC	
Spl_mentor_specs	SPLM	
Spl_survent_specs	SPLS	
Spss_specs	SPSS	
Sum_file	SUM	
Text_hold_size=	TEXHOLD=	
Region	R	17
Remove	REM	4
Rename	REN	31
Repeat	REP	31
Repeat_vars_alpha_only	REPVARA	31
Replicate(	REP(	20
Reset (question type)	RSE	7
Reset (tilde command)	RESET	11
Read_cases	READCASE	
Reset_column	RESETCOL	5
Response_on_right	RESPRIGHT	3
Restore	RESTORE	11
Restore_column	RESTORECOL	5
Resume	RES	2
Resume_here	RESUMEH	2
Resume_where_at	RESUMEW	2
Rft_cat_01	RFTCAT01	5

Item	Abbreviation	Reference
Rft_cat_punch	RFTCATP	5
Rft_cat_response	RFTCATR	5
Rft_cat_spread	RFTCATS	5
Rft_on	RFTON	5
Rft_save_loops	RFTSAVEL	5
Rft_unwind_loops	RFTUNWINDL	5
Rotate	ROT	2
Run_label=	RUNLAB=	31
Safe_specs	SAFESPEC	1
Save_as_db	SAVEASDB	31
Save_as_file	SAVEASFILE	31
Save_column	SAVECOL	5
Save_keys	SAVEKEY	31
Screen_lines=	SCREEN=	1
Select_value(	SELECT(	20
Set	SET	11
Allow_edit_change	EDITCHANGE	
Allow_multiple_weight_stats	MULTIWT	
Automatic_new_line	AUTONEWLINE	
Automatic_new_page	AUTONEWPG	
Automatic_reset	AUTORSET	
Automatic_tables	AUTOTAB	
Begin_table_name=	BGNTAB=	
Bottom_margin=	BOT=	
Case_sensitive	CASESENSITIVE	
Clean_allow_blanks	CLNAB	
Clean_error_number	CLNERRNUM	
Cleaner_definition=	CLNDEF=	
Column_repeat=	COLREP=	
Column_repeat_override	COLREPOVERRIDE	
Confirm_blanked_columns=	CONFIRMBLANK=	
Delimited_tables	DELIMTAB	
Banner	BAN	
Column_width	CWID	
Delimiter=	DELIMTER=	

Item	Abbreviation	Reference
Delimited_mode	DELIMMODE	
Do_table_name	TABNAME	
Footer	FOOT	
Header	HEAD	
Labels	LABELS	
Stub	STUB	
Stub_width	SWID	
Title	TITLE	
Title_2	T2	
Title_4	T4	
Title_5	T5	
Double_space_errors	DOUBLESPACEERRORS	
Drop_banner.	DROPBAN	
Drop_base	DROPB	
Drop_column	DROPCOL	
Drop_column_short_weight	DROPCOLSHORTWT	
Drop_column_weight	DROPCOLWT	
Drop_filter	DROPFIL	
Drop_footer	DROPFOOT	
Drop_header	DROPHEAD	
Drop_local_edit	DROPLOCEDIT	
Drop_row	DROPROW	
Drop_row_short_weight	DROPROWSHORTWT	
Drop_row_weight	DROPROWWT	
Drop_stat	DROPSTATS	
Drop_stub	DROPS	
Drop_stub_preface	DROPSTUBPREF	
Drop_stub_suffix	DROPSTUBSUF	
Drop_table_set	DROPTABSET	
Drop_title	DROPT	
Drop_title_2	DROPT2	
Drop_title_4	DROPT4	
Drop_title_5	DROPT5	
Drop_weight	DROPW	
Echo_tables	ECHOTAB	

Item	Abbreviation	Reference
Edit_dump	EDITDUMP	
Edit_run_chain	EDITRUNCH	
Enhanced_default_titles	ENHDEFT	
Error_limit=	ERRLIM=	
Error_process	ERRPRO	
Error_review	ERRRE	
Error_stop	ERRST	
Errors_to_print_file	ERRPRT	
Error_summary	ERRSUM	
Ignore_suppress	IGNORESUP	
Leave_room=	LVROOM=	
Logging	LOG	
Loopkickout	LOOPKICKOUT	
Maximum_past_cases=	MAXPASTCASE=	
Max_statistics_size=	MAXSSIZE=	
Max_variable_size=	MAXVSIZE=	
Mean_statistics_only	MEANSTATONLY	
Next_column=	NEXTCOL=	
Page_length=	PGLEN=	
Page_number=	PGNUM=	
Page_number_increment=	PGNUMINC=	
Page_width=	PGWID=	
Print_all_errors_stop	PRTALLERRST	
Print_specs=	PRTSPEC=	
Print_tcon	PRTTCON	
Procedure_dump	PROCDUMP	
Production_mode	PROD	
Region_coding_mode=	REGIONCOD=	
Reset	RESET	
Save_table=	SAVETAB=	
Sequential_read	EQUENTIALREAD	
Show_memory	SHOWMEMORY	
Stack_1=	STK1=	
Stack_2=	STK2=	
Start_tcon	STARTTCON	

Item	Abbreviation	Reference
Statistics_base_AR	STATBASEAR	
Statistics_dump	STATSUMP	
Status	STATUS	
Table_drop_mode=	TABDR=	
Table_drop_warn=	TABDRWARN=	
Table_dump	TABDUMP	
Table_effort=	TABEFFORT=	
Table_field=	TABFLD=	
Table_lines=	TABLINE=	
Table_missing_mode=	TABMISS=	
Table_modify_mode=	TABMOD=	
Table_name=	TABNAME=	
Table_number_adjustment	TABNUMADJ	
Table_present=	TABPRES=	
Table_set_match_error	TABSETMATCHERR	
Table_set_match_warn	TABSETMATCHWARN	
Table_store_mode=	TABST=	
Testing_mode	TEST	
Top_margin=	TOP=	
Training_mode	TRAIN	
Unweighted_top	UNWTTOP	
Variable_name	VARNAME	
Work_room=	WORKROOM=	
Zero_fill	ZF	
Set_quota.	SETQUO	4
Show	SHOW	11
Show_core_farmark	SC	31
Show_defines	SHOWDEF.	31
Show_key=	SK=	31
Show_question_labels	SHOWQLAB	2
Show_question_labels	SHOW	1
Sort	SORT	11
Ascending=	A=	
Break_variable=	BV=	
Descending=	D=	

Item	Abbreviation	Reference
Shut_down_console	SHUTDOWNCON	
Skill_level	SKILL	
Spec_files	SPEC	11
Line_length=	LINELEN=	
Spec_rules	SPECRULE	11
Base	BASE	22
Base_comment	BASECOMMENT	22
Cln_check	CLNCHK.	22
Column_mean=	MEAN=	22
Do_loops	LOOP	22
Justify=	JUSTIFY=	22
No_base	NOBASE	22
Store_tables	STORETAB	22
Stub_default	STUBDEF	22
Table_set	TABSET	22
Use_print_enhancements	USEPRTENHANCE	22
Spec_width=	SPECWID=	1
Special (block)	SPECIAL	2
Special (Information)	SPC	7
Spl_data_locations_ok	SPLOK	1
Square_root(	SQRT(	20
Standard_deviation(	STD(	20
Statistics	STAT	22
Status	STATUS	31
All	ALL	
Db	DB	
Files	FILES	
Input	IN	
Output	OUT	
S_time	STIME	31
Stop_watch	SW	21, 31
String	S	17
String_from_number(	STRINGFROMNUM(	20
String_length	STRLEN(	20

Item	Abbreviation	Reference
Strip(	SRIP(	20
Study_name	STUDY	31
Subscript(	SUBSCRIPT(	20
Substitute(	SUBSTITUTE(	20
Sum(	SUM(	20
Summary_file	SUM	1
Suspend	SUS	1
Suspend	SUSP	2
System	SYS	31
Table_from_numbers(	TABFRNUM(	20
Table_name	TABNAME	18
Tab_warn	TABWARN	31
Target	TARG	4
Terminal_pause=	TERMPAUSE=	31
Text	TEX	6
Text_area_status	TEXSTATUS	8
Text_help	TEXHELP	1
Text_start=	TEXT=	1
Time_stamp	TIMESTAMP	1
Time_zone=	TZ=	1
To	TO	21
Total	TOTAL	8, 21
Translate	TRANS	11,31
Ascii	ASCII	
Binary	BIN	
Cards_image	CARD	
Dots	DOT	
Dta	DTA	
Hex	HEX	
In	IN	
No_directory	NODIR	
Out	OUT	
Stop_after	STOP	
Swapped_binary	SWAPBIN	

Item	Abbreviation	Reference
Swapped_hex	SWAPHEX	
Uncompressed UNPRESS		
Triple_quotas	TQ	1
True	TRUE	18
Tsig(	TSIG(	20
Undefine	UNDEF	31
Update	UPDATE	11
Values(	VALS(	18
Variable	VAR	6
Variable_exists(	EXIST(	20
Vector	V	17
Vector_function(	VFUNC(	20
Version	VER	31
View	VIEW	11
View_execute	VIEWX	4
View_quota	VIEWQUOTA	2
Wait	WAIT	31
Word_matches(	MATCH(	20
Word_starts(	START(	20
Work_start=	WORK=	1
Write_specs	WRITESPEC	11
C_mentor_specs=	CMentor=	
C_specs=	CSPECS=	
Persee_specs=	PERSEE=	
Quantum_specs=	QUANTUM=	
Spl_mentor_specs=	SPLM=	
Spss_specs=	SPSS=	
Table=	TAB=	
Text=	TEXT=	
X(	X(	20

This is another reference table for Survent and Mentor:

Survent		Mentor	
1	Study Header option	11	Tilde commands
2	Interview Control (3.2.1)	12	CLEANER keywords allowed in a procedure
3	Compose Control (3.2.2)	13	CLEANER keywords not used in procedure
4	Compile Control (3.2.3) Survent	14	DEFINE main keywords
5	Data Control (3.2.4)	15	Print #information# in printed text
6	Data Entry Question Types (2.4)	16	Procedure controls
7	Control Statements (3.1)	17	Variable types
8	Composing and compiling questions in PREPARE (2.3.1)	18	Constants
9	The PHONE Statement (6.1.1)	19	Objects
		20	Functions
		21	Region keywords
		22	Spec generation control (4.7)
		31	Meta

Appendix C
Glossary

GLOSSARY OF CFMC TERMS

Abort To terminate abnormally.

ABORTJOB An HP MPE command used to stop a process (job). See Stream Job.

Alias A name or label used as an alternative means of referring to someone or something, in Survent, used to reference the answer given to a previous question. For an example, see Appendix Z of the Survent manual. See Back-reference.

All Possible Pairs (Mentor) Statistical test that is a significance test between-multiple means or percents. Tests all pairs of columns individually. Most likely of all Mentor statistics tests to find columns significantly different. See NEWMAN-KUELS, Fisher.

Ampersand In To call the contents of another file into the current file. This is done by putting an ampersand symbol in front of a file name, for example "&file2".

ANOVA (Mentor) Statistical test that is a significance test among multiple means.

ANOVA Scan (Mentor) Statistical test that is a significance test among multiple means, that is a two step process to quickly determine if there is a difference between multiple means. A refinement of the All Possible Pairs test.

Answer Array (Survent) An area in memory where Survent keeps track of answers so they can be displayed later in an interview. See alias and back-reference.

ASCII Pronounced "askee"; acronym for American Standard Code for Information Interchange. A coding scheme that assigns numeric values to letters, numbers, punctuation marks and special characters. ASCII files are generally transportable across all computer systems. ASCII files are sometimes called a text files, text-only files or flat files.

Autodialer Hardware that opens a telephone line and dials a stored telephone

number.

Auto Fixing (Mentor) A type of cleaning method. Done in batch mode, when cleaning errors are encountered they are automatically fixed based on dictates in the cleaning specifications. See Data Lookup.

Awareness Grid A list of items that are repeated for a series of questions. Often a list of brand names, and a record is kept for the number of times each of the brands is mentioned.

Axis The vertical or horizontal variable in a cross-tabulation table.

Back-reference (Survent) Showing the answer to a previous question.

Bang Computer slang for the exclamation point(!). If tech support tells you to "type bang dataloc," they mean "[!5/10.2]".

Banner The text at the top of a table, made up of the headings for each of the columns.

Banner Point The headings for columns in a table. All the banner points together make up the banner at the top oSurventf a table.

Base Subset of a total sample. Mentor allows you to exclude respondents from a table. Often used as a verb, as in "Do you want to base this table?"

Batch File A file that contains a list of commands that, when executed, carries out the commands without user intervention.

Binary (1) The numbering system based on twos (just as the decimal system is based on tens). Numbers are represented using only the digits 0 and 1. (2) IBM360 column binary coding which is the format for CfMC System files. Also see SWAPPED BINARY. (3) When data is stored as multiple punches in each column, also called multipunched data.

Block Commands See Tilde Block.

Blow Error Errors that occur when Survent terminate abnormally or "blow up." See a list of Blow Errors in the Survent manual, Appendix D.

Boolean A condition or variable that returns either TRUE or FALSE. In Mentor, the Boolean joiners are AND, OR and NOT. (Also called logical joiners.)

Bottom Box See Top Box/Bottom Box.

Break Table (Mentor) A table that splits one long variable into a series of rows or columns. Different from regular tables, Mentor writes the data vertically for a defined rows (horizontally for columns) until it reaches a break command and then wraps to the top of the next column to continue writing data. This type of table is often used to put information that would appear on separate tables onto one page.

Bucket An area to hold information. In Survent, phone numbers with the same calling schedule are called stacks. Stacks of phone numbers are sorted into buckets.

Callback When an interview has been interrupted and the interviewer has scheduled a time to call back and complete the interview. Also known as a scheduled callback. See Suspend.

Call Disposition See Call Status.

Call History (Survent) Information about a phone call for a verbal questionnaire, such as the call's start and stop times, the interviewer ID and its status.

Call Status (SurveSurventnt) A status code for a phone number stored in the phone file. Each time Survent gets a phone number, it will return the number to the phone file with a call status, such as resolved (completed interview, non-working number, etc.) or call back (no answer, busy).

CAPI Computer Assisted Personal Interviewing, face-to-face interviewing.

Card See Record.

Case An individual set of data from a data file; a respondent.

Case Header (Mentor) A piece of each data case that holds case information, such as the internal case ID. If you want to change a record's case ID, you must change BOTH the data in the columns for the assigned case ID and the internal case ID in the case header.

Case ID A record identification number, usually used to distinguish respondents. CfMC data files have both an assigned case ID and an internal case ID. The assigned case ID is usually sequential and often stored in the first four columns of data of a record. See Case Header.

Case Flag A flag is a marker used by a computer in processing information. In Mentor, case flags are set for deleting, cleaning and updating cases. An error flag is set by any CHECK, CLEAN, EDIT, or ERROR statement that is true for that case.

Category A specifically defined division of the data field. A category can represent a single group of respondents or be “netted” to represent more than one group.

CATI Computer Assisted Telephone Interviewing, what Survent does.

Cell The intersection of a row and a column. In a Mentor table, a cell shows the respondents who satisfied the conditions for inclusion in a table. Cells can be printed as frequencies and/or percentages.

Character One symbol of a set of elementary symbols. The set usually includes letters A-Z, digits 0-9, and other various special symbols.

Chi-Square (Mentor) Statistical test that tests for independence. A cross-tabulation test. Tests the significance between parts of a column (banner points) with parts of a row(stub). This test is often used when means are not applicable.

Cleaning To validate the integrity of data based on user defined instructions. Can be done interactively or in batch mode.

Cleaning Specs A batch file that has a list of commands to clean data.

Codebook The layout of data locations for a questionnaire.

Code List A list of categories created from a list of open-ended responses.

Coding (1) The process of categorizing the answers from open-ended questions. Results in a code list. (2) The act of programming.

Coding Mode (Survent) A type of existing case mode, that is, Survent picks up a completed cases in a data file and allows you to verify and/or modify them. This requires that you write a Survent questionnaire that interrogates a data file. See Existing Case Mode.

Column When used generically, it refers to an individual category in the vertical axis. When used as a keyword, it refers to the data definition for the entire vertical axis. It can also refer to a data location where the answer, or part of the answer, for a particular question is stored.

Comment Out To temporarily disable one or more lines of code from a program by proceeding them with by a comment symbol. In Survent or Mentor, you can comment out a single line by putting two apostrophes(") at the beginning of line, or in Mentor, you can comment out a whole tilde block by placing a minus sign after the tilde and before the command (e.g. ~-DEFINE).

Compile To translate a high level code to a lower-level, machine-readable format. The PREPARE module compiles a questionnaire specification file (usually with the file extension QPX) into a file that Survent can read (with the file extension QFF).

Compiler Command (Survent) Commands that control questionnaire functions. Compiler commands are in enclosed in curly braces and preceded with an exclamation point, e.g. {!ALLOW_SUSPEND}.

Condition The state of an expression or a variable--for example, when a result can be either true or not true or equal or not equal. IF conditions are used in data cleaning and table building to cause an action. For example, IF the answer to question six is "yes," then check to make sure the answer to question seven is one, two or three.

Confidence Interval An estimation of the range of where the mean will fall with repeated sampling. A 95% confidence interval means that, under the assumption of normality, 95% of the time the mean will fall within that range.

Constant See System Constant.

COSI A software program that creates statistical table cross-tabulations and graphs. Runs under Windows.

Control Statement (Survent) A question type that does not prompt for a response from the interviewee, but rather controls the interview process, such as displays information to the interviewer or does internal data generation. See Data

Entry Statement.

Cross-Tabulation An action that produces a table that reflects the intersections of two variables.

Data Entry Statement (Survent) A question type for collecting data, it displays a screen and prompts for a response. See Control Statement.

Data Variable A definition of the type of data in the field specified and its categories. Always contained inside square brackets [ ].

Data Layout The location, in rows and columns of data in a data file. You can hard code, that is, specify the data locations for the answers for a questionnaire or let Survent do the data layout for you.

Dataloc Data location. In brackets, written in a card/column.column width format. E.g. [1/10.3] Indicates card one, column ten for a width of three, i.e. columns ten, eleven and twelve.

Data Lookup (Mentor) A type of cleaning method. Done interactively, when you encounter a cleaning error, you go back and look up the answer on the original (paper) survey. See Auto Fixing.

Day Parts (Survent) The times of day the phone system uses to schedule calls.

DB Entry An item stored in the DB(database) file. When Mentor builds tables, it generates and stores information such as data definitions as DB entries for future use.

DB File The database file. A machine-readable file generated by a Survent or Mentor compilation that stores variables for future use. It can also store labels, data cleaning and generating procedures, and tables themselves.

Default A word used as both noun and verb in reference to the choice made by a program when the user does not specify an alternative.

Delimiter A character that separates items in a data file. Mentor can allows you to in input or output delimited data.

Delimited Data Data that is separated by a delimiter, often from a database or spreadsheet program. In the following example, commas separate the fields in a database record:

```
Jones,801 Haight Street,San Francisco,CA,94117.
```

Demo Deck A data file that contains general information about respondents, usually has an ID number field in common. Like a deck of cards, it can be shuffled or sorted.

Dialing Parameters See Shop File.

Directory A place where files and other directories are stored on disk. In a hierarchical file system, the topmost directory is called the root directory. Directories within directories are called sub-directories. In the Windows, X-windows and

Macintosh operating systems, directories are represented graphically by file folders.

Disk-based Response List (DBR) (Survent) When a response list is stored as a separate file. This allows for large response lists, up to 10,000 categories.

Disk-based Recode Table (DBR) (Mentor) When a response list is stored as a separate file. This allows for large recode tables, up to 10,000 categories.

Dot Computer slang for period(.). If tech support tells you to "Type star dot star," they mean "*.*"

Dotting When periods (dots) appear sequentially on a computer screen, usually to indicate that a process is running. This is used when the SERVER module of Survent is running, and when Mentor is processing data.

DTA File A CfMC data file from an older SPL version of the software.

End-of-File (EOF) A code placed by a program after the last byte in a file. An EOF character is a marker that tells the computer's operating system that no additional data follows.

Environment Variable Settings for specifications for the software, such as the command path (where to look for files). Use the SET (DOS) or ENV (UNIX) commands to see how the environment variables are currently set.

Existing Case Mode (Survent) When you write data out to or modify existing cases in a data file, opposed to interviewing mode. See Coding mode, View mode and Phone Sys Data Rec mode.

Expression (Mentor) A complex variable made up of two or more variables. Joiners connect variables to form expressions. See Joiner.

Field A location in a data record in which a particular type of data is stored.

Filter (Mentor) A variable that is used in table specifications to only include a subset of responses. A filter runs across a whole study, while a base is for individual tables.

Fisher (Mentor) Statistical test that is a significance test between multiple means (analysis of variance). A refinement of the All Possible Pairs test.

Flag A marker used by a computer in processing information. See Case Flag.

Flat File See ASCII.

Folder See Directory.

Footer Text that prints at the bottom of each page. A table-building option in Mentor.

Function (Mentor) A type of joiner used to get special values or translate one type of element to another. The five function types are: arithmetic, vector, number-returning, logical, table-related, integer and string. See Chapter 8 of the Mentor manual.

Gen A process that generates something. In Survent, GEN is a statement and a question subtype, often used to generate multiple questions using the same question list, but requiring different responses depending on a prior response.

Global Scratch See Scratch. Grid Question (Survent) Having more than one question on the screen at a time, the respondent can answer in any order.

Group A place where files are stored on disk in the HP MPE operating system. Equivalent to a directory or folder.

Header Text that prints at the top of each page. A table-building option in Mentor.

Hierarchical Processing See Master-Trailer Processing.

Hexadecimal(Hex) The numbering system based on 16 digits. These include 0-9 and A-F.

Holecount The frequency of punches for designated columns. Can be generated by Mentor specifications or the HOLE utility.

ID See Case ID.

Inbound Dialing When the respondent initiates the call to a phone center, as opposed to outbound dialing.

Indexed Phone File (Survent) A phone file that is organized for easy for easy retrieval. Often used for inbound dialing.

Indexing A way to sort a data file for quicker accessing of information. Results in an indexed data file. Mentor allows you to select items or sections of data for faster access with the ~MAKE_READ_CONTROL command.

Interviewer ID (Survent) This is used to identify the interviewer in the data; has to be four characters or less.

Interviewing Mode (Survent) The normal operation of Survent, that is, a new case and case ID is generated and written to the data file when a case is completed. See Existing Case Mode.

IPC file (Survent) A file used by networked devices that serves as a mailbox for internal network messages. Each network entity, such the SERVER and each interviewing station has their own IPC file. You can check for the existence of these files, but cannot read them.

Joiner (Mentor) Commands used to connect variables to form an expression. Often used to put data from different data locations(questions) on the same row. Joiners can be relational operators, arithmetic, axis, vector or Boolean.

Key A small piece of hardware that attaches to a PC's parallel or serial port that allows you to run Survent or Mentor. Used to prevent unauthorized distribution or use of CfMC software.Survent

Keyword Options to tilde commands that have their own options.

Kick To increment by one. Also see Page Kick.

KRUSKAL-WALLIS (Mentor) Statistical test that is a significance test between two rows. A variation of a rank-order test. This test only works on independent rows. A non-parametric order test. Can be used for rating scales, but treats ratings as ordinals rather than values (i.e. 1 means 1st rather than the value one).

Label The name given to variable definition, used for future reference. In Survent, you can assign a label to a question so you can refer to it later in the interview or in Mentor specifications.

LDEV (Logical Device) A numeric identifier that the HP MPE operating system uses for devices, such as terminals or printers. Each external device is assigned a unique logical device number.

Likert Scale Any scale of polar opposites, classically:

```
(1) Excellent
(2) Very Good
(3) Good
(4) Fair
(5) Poor
```

List File Where the output (error messages, tables, etc.) goes from a Mentor (or Survent) run. The list file can go to a file, or directly to a printer, or the screen.

Local Scratch See Scratch.

Loop Any process or series of variables that repeats.

Loop Variable (Mentor) A type of variable that looks at data in multiple locations.

Overlay tables are a report of loop variables. See Overlay Table.

Marginal Another name for a holecount. See Holecount.

Master-Trailer Processing Table generation for a type of study where a primary (or master) questionnaire collects general information and then secondary (or trailer) questionnaires collect additional information. For example, household data is collected in the master questionnaire and then questions for each person in the household is collected in a secondary questionnaire. Tables from this type of study can then display information based on households, individuals, or a combination of information from both groups. Also called Hierarchical Processing.

Mean Summary Table A table that shows the average response to a question or questions, usually used when you have many scalar questions with the same response set in a group. (Also known as Summary of Means Table.)

Meta Command High-level commands, most that work in both Survent and Mentor. They are preceded with the greater than (>) sign. Meta commands that are specific to Survent are defined in the Survent manual, and the rest are defined in the Utilities manual.

Monitor (1) (Survent) Person running SURVMON, a utility that allows monitoring of interviews from outside SURVSUPR. (2) A computer screen. Sometimes called a display.

Multipunch When multiple answers appear in one column of data. Also called binary or punch data.

Nest To insert a command between the beginning and end of another command. A nested loop is when a loop is completely contained within another loop. Commands that are often nested are REPEAT, ROTATE and IF-THEN-ELSE.

Net A total. Often used in table building for respondents who said at least one of a group of responses. Subsets of nets are called subnets.

NEWMAN-KUELS (Mentor) Statistical test that is a significance test between means (analysis of variance). This test estimates which of the two columns are most likely to be different and tests them first. Less likely to find significance that ALL POSSIBLE PAIRS test. See All Possible Pairs.

Numbered Quota (Survent) One of the three types of quotas. This method uses numbers rather than quota names. If you have over 450 quotas, you must use numbered quotas. You can have up to 32,000 numbered quotas. See Quota.

Numeric Distribution See Frequency.

On the fly The ability to change something while a program is running. For example, Survent allows you to change the language questions and answers are displayed in on the fly.

Operators Relational that which compares two variables or a constant with a variable or a function with a variable: >, <, <=, >=, =, <, >. Logical - connectors of variables or locations; also called Joiners.

Open-ended Question A survey question that does not provide a list of answers to choose from. See verbatim.

Overflow Error See Stack Overflow.

Overlay Table (Mentor) A table that looks at data from multiple locations. It shows multiple responses from individual respondents. See the Advanced Tables chapter in the Mentor manual.

Page Kick The character or sequence of characters that tells a printer to eject a page.

Parameter A value given to a variable. It can be text, a number, or an argument name assigned to a value that is passed from one routine to another.

Path The route followed by an operating system to store and retrieve files on a disk. In a hierarchical filing systems (DOS,UNIX), the path starts at the root directory and then lists all the subdirectories that lead to the file. For example, the following indicates that the file myfile.spx is located in a directory called Mentor,

which is located in a directory called cfmc.

```
\cfmc\mentor\myfile.spx
```

Phantom File (Mentor) A dummy input file used to test table specifications.

Phone File (Survent) A file that contains a record for each phone number, plus the call parameters copied from the shop file.

Phone Sys Data Rec Mode (Survent) A type of existing case mode, that is, Survent picks up a case that already has ID information in it and then writes interviewing information to it when the case is complete. This requires that you write a Mentor job that creates a "blank" case for each phone number in the phone file. See Existing Case Mode.

Platform The combination of hardware and operating system that software runs on, often just identified by the operating system. CfMC software runs on the DOS, SCO UNIX, and HP MPE-IX platforms.

Point Scale A numbered list of answers ranging from high to low, often (1)excellent (2)good (3)fair (4)poor. See Likert Scale.

Point Scale Reversal (Mentor) A process to reverse the meaning of numbers in a point scale. This is done because during interviewing, "excellent" comes first in the point scale and therefore the lowest number on the point scale, and in tables, "excellent" is associated with the highest number on a point scale.

Prepare A module of CfMC software(both Survent and Mentor) that compiles questionnaires and creates files .

Print File (Mentor) An ASCII file that contains the output generated by a Mentor run (usually tables). You can also direct error messages to the print file. See List File.

Proc A group of commands that you want a program to execute. In Mentor, PROCEDURE= (or PROC=) is a an option to the ~DEFINE command that allows you to perform some operation on the every case in a data file or some specified subset of the data file.

Prompt A character, symbol or group of characters that appear onscreen, indicating that the program or operating system is waiting for your input. In DOS, "the C prompt" means the letter of the disk drive appears on screen (as "C:") and the operating system is waiting for your command.

Punch See Multipunch.

Punches The possible values for a column: 1-9, 0, X, and Y.

QPX file (Survent) A questionnaire specification file, usually given a file extension of QPX. The PREPARE module compiles a QPX file into a file that Survent can read (with the file extension QFF).

Quota (Survent) Counters that are kept across interviews (or interviewers) that are usually used to determine whether to continue an interview based on a set

of questions asked at the beginning of the interview. Survent has three types of quotas: standard, triple and numbered.

Ranking A way to organize rows of a table into a hierarchy. Done by assigning rank level to stubs. Includes the use of nets and subnets.

Rating Scale See Point Scale.

Recode A Survent utility that lets you add, change or recode data in a CfMC data file.

Recode Table A table showing response text (answers) and their associated codes or punches. Also called a Response List.

Record 80 columns of data. Each data case can be divided into 80 column records. By default, Mentor references a data field by its record number, column and width. Multiple records from one respondent is called a CASE..

Reformat (1) In Survent, a utility that reads a questionnaire file (QFF) and data file (TR) to produce an ASCII data file. Sometimes called "spread data" because it takes multipunched answers in a single column and spreads them across multiple columns. (2) In Mentor, any way you manipulate data across all cases.(3) In data storage, to prepare for reuse a disk that already contains data, effectively destroying the existing contents. (4) In word processing to change the look of a document by altering stylistic details.

Relational Operators See Operators.

Relops See Relational Operators.

Respondent One person who answered a set of questions; their set of answers makes up one data case.

Response List See Recode Table.

RDG (Random Data Generator) (Survent) An option used to generate random responses to questions, used to test questionnaires and create dummy data files.

RPG (RePort Generator) (Mentor) Using the PRINT_LINES command to generate specialized reports from a data file, rather than tables.

RFT See Reformat.

Row (1) Generically, an individual category in the horizontal axis. (2) In Mentor, a table building command that refers to the data definition for the entire horizontal axis.

Run The execution of a program.

Sampling Gathering data from a representative subset of a larger group (called a population).

Scale See Likert Scale or Point Scale.

Scalar Table A table that shows the answers to a point scale question or questions.

Scratch Generically, a memory region or a file used by a program as place to keep work in progress. Survent uses both a local scratch and a global scratch. Local scratch is an area in memory that holds information between interviews for a particular interviewer. Global scratch is a place in the quota file used to hold information other than quota numbers. Mentor allows for a scratch area at the end of each record.

Server On a local area network(LAN), a computer running administrative software that controls access to resources such as disk drives or printers.

SERVER A Survent module that oversees the communications among networked interviewing stations.

Shell A piece of software that provides communication between the user and the operating system. The Macintosh Finder is a shell, and COMMAND.COM is an MS-DOS shell. UNIX shells include Borne shell (sh), Korn shell (ksh) and the C shell (csh).

Shell Script A file that is executed by the shell of an operating system. The term is usually used to refer to scripts that are executed on UNIX systems.

Shop file (Survent) A file used by the Survent phone system. It contains the initial parameters used to dial phone numbers, such as time zones, times available to call, and number of calls to make.

Skip Pattern In interviewing, to set up a path, depending on how a person answers a question, for which question they get asked next (and which questions get skipped). Mentor allows you to set up cleaning specifications to check that there are not answers on questions that should have been skipped. This is also called "verifying skip logic."

Sort Key A field (commonly called a key) used to sort a data file. The COPYFILE utility can sort data files on up to five different sort keys.

Spec File A file that contains the commands for executing a procedure in batch-mode.

Spread Data See Reformat.

Squiggly CfMC slang for a curly brace. ({)

Stack (Survent) A group of phone numbers with the same calling schedule. See Bucket.

Stack Overflow An error that occurs when there is problem with an area of memory. This occurs when Mentor has more data than columns to put it in, and then the columns in the data file will be filled with asterisks. Or, software or the operating system can run out of memory. This problem usually requires that you change the configuration of your computer (add memory and/or change configuration files).

Standard Quota (Survent) One of three types of quotas. See Quota.

Star Computer slang for asterisk (*). If tech support tells you to "Type star dot star," they mean "*.*". Sometimes called splat.

Station Number (Survent) A code used to identify the personal computer or terminal used for an interviewing session.

Statistical Significance The confidence with which you can state that your results could not have happened randomly. Consult a statistics textbook for a discussion of this concept.

Statistical Summary Table A table that shows summary data for several questions, rather than frequencies for individual questions.

Stream Job A process (job) running on an HP MPE computer. From the MPE STREAM command. See ABORTJOB.

String A sequence of letters, numbers and symbols.

String Variable An arbitrary name assigned to a string of characters so the contents of the string can be called by just using the variable name.

Stub The text on the vertical axis (rows) of a table. Usually the answers to questions, either a response list item or a numeric range.

Student Newman-Keuls (SNK) See Newman-Keuls.

Study Code (Survent) The name assigned to as study. Also called a Study Header.

Study Server See SERVER.

Subnet See Net.

Summary Table See Mean Summary Table.

Supervisor See SURVSUPR.

SURVSUPR A control program for Survent interviewing.

Suspend When a respondent wants to stop in the middle of an interview. Usually, the interviewer schedules a time to call back and complete the interview. See Callback.

Swapped Binary A type of file where the first and last bits are swapped. You can recognize this type of file when seeing sixes and sevens where you would expect to see zeros and ones (binary punches 01-9XY are swapped binary punches 6-9YX012354). If you transfer a standard IBM-360 column binary file from the HP3000 to the PC, you should treat it as a swapped binary file.

System Constant (Mentor) Commands that allow you get current information from your computer or from data files, such as the case number, the date or the number of errors for a particular run. Mentor has three types of constants: variable, case-reading, and system information.

System File A specially formatted CfMC file that holds data. In a system file, one record is the length of an entire case, no matter how many columns of data are

in the case. In addition, it has a special segment (called the case header) in each record that holds the internal case ID and flags for deleting, cleaning and updating the case. System files have an extension of TR.

T-test (Mentor) Statistical test that is a significance test between two means (analysis of variance). See Z-test.

Table The format by which the tabulated data is presented, usually a two-axis rectangle of numbers.

Table Title Text that describes what is in a table. In Mentor it prints below the table heading but before the banner text.

Tabulation The process that utilizes computer technology to prepare, summarize and present data that has been collected into a table format. The function of Mentor.

Tab Specs A batch file that Mentor uses to generate tables, perform statistics and format results.

Table Set (Tab Set) (Mentor) A section of Mentor specs that defines the elements of an individual or group of tables, such as the banner, header, column and row. Typically, you will have a global tab set for the elements that are the same across all (most) tables, such as the banner and headers, and individual tab sets for each question, which provides the title (the question text), stubs (the answer text)and row definitions (data location of the answers). See the Mentor command ~DEFINE TABLE_SET.

Text Area Where data from Survent text questions is stored. It is stored in format compressed at the end of the data file.

Text Pointer A pointer is variable that contains the memory location of some data rather than the data itself. Survent and Mentor use pointers for the answers to text questions.

Text Variable A definition that specifies table text labeling, preceded by a command to identify where it will be used in the table, e.g. HEADER= .

Tilde Block Some Mentor tilde commands when issued start a section of the program that allows you to enter a group or block of options and sub-options without issuing the tilde command again. This is known as a tilde block. Tilde block commands are CLEAN, DEFINE, and SET.

Tilde Command Mentor commands that start with the tilde(~) symbol. Tilde commands are listed in Appendix B of the Mentor manual.

Topline Report A tally of closed-end question responses.

Top Box/Bottom Box A row on a table that is a total of the highest (top box) or lowest ratings (bottom box) for a brand or attribute. This row is in addition to each individual rating. Usually combines the top two or bottom two categories, for example "excellent" and "very good" are combined into a row labeled "top box."

TR File See System File.

Triple Quota (Survent) One of three types of quotas. This method uses three separate counters for each quota, and allows you to change target values easily. See Quota.

Upkick Computer slang for caret(^). If tech support tells you to "Type two upkick one," they mean "2^1".

Utilities CfMC menu-driven programs used to accomplish repetitive tasks, such as sorting data files, or creating basic reports such as holecounts.

Variable A named storage location capable of containing a certain type of data that can be modified during program execution.

Verbatims The responses to open ended questions in which the responses have been recorded word for word, or verbatim.

Vector An axis; a group of categories.

Vector Joiner (Mentor) A type of joiner that connects multiple categories. Vector joiners are WITH, BY, WHEN, INTERSECT, NET, and OTHERWISE.

Vertical Axis On a table, a row that contains a stub and row data.

View Mode (Survent) A type of existing case mode, that is, to look at completed cases in a data file to verify and/or modify them. This is done with the SURVIEW utility. See Existing Case Mode.

Wave Study A study that interviews the same people or asks the same questions of different people over time. Each reiteration of the study is called a wave.

Weight An assigned value that changes the value of a frequency in proportion to a sample. Weighting a table means assigning a weight to each answer, often based on assumptions about a population.

Work Area (Survent) A user-specified area in a data file used to place data from an interview that is kept separate from the rest of the data. Often used if you want to add additional questions or control statements to a questionnaire without changing the existing data layout.

Z-test (Mentor) Statistical test that is a significance test between two means (analysis of variance). This test assumes a normal population. A variance of the T-test.

Zone Table (Survent) Used by the phone system, a file that has area codes and exchanges assigned to time zones across the U.S.

Appendix D
CfMC Conventions

Comment Out (‘ ’)

Two single apostrophes are used to “comment out” the remaining portion of a line. All entries from the two apostrophes to the end of the line will be ignored by the program, but will print out in your program listing for those programs that list their specs (Mentor, for example). This must be two single apostrophes, not one double quote. Only one apostrophe is needed if it appears in column one.

Spec file

Programs containing a SPEC FILE prompt allow that particular program to be run either from an interactive session or as a batch job (i.e., Mentor, DBUTIL). If the user plans to enter each line individually, while the program is prompting, only Enter is pressed when the screen displays the message:

```
SPEC FILE-->
```

By entering one program statement at a time, the program can check the syntax of each line as it is entered, and will display an error message and request for re-entry of the command if a mistake has been made. It is also possible to create a spec file that will hold all or part of the desired program commands.

This is useful if the user is very familiar with the syntax necessary on all program statements and needs to create a long and complex program run. Statements can be entered and checked before ever running the program, and this allows the repeated rerunning of the program, without wasting time re-entering many commands. A spec file is created in the manner of any other editor file.

The spec file is the file containing questionnaire or tabulation specifications along with other commands. It must be an ASCII file, with a line length of 5000 characters or less. It must not contain format-control characters often created by word processors. If it contains tabs, a warning will print at every line with tabs unless you precede the lines with a >-TAB_WARN command. The program reads the file

line by line.

If you have a file, you would enter the file name, or the drive and directory or group and account) of the file if they differ from your current drive and/or directory, with the file name.

The syntax for the file name is:

```
Drive letter:\Directory1\SubDirectory1\ filename.Extension
(DOS)

/Directory1/SubDirectory1/ filename . (UNIX)

filename.group.account (MPE)
```

The drive letter is first, followed by a colon (:). The default is the drive of the computer you are currently signed on to. If the file is located in a directory and/or subdirectories other than the one you are signed on in, enter the directories separated by backslashes (\). Then comes the file name, which must be from one to eight characters long, starting with a letter.

The extension is a naming convention that is up to you.We suggest you use an extension on the file name to specify the type of file it is. The file name extension can be from one to three characters (preceded by a period in DOS). The only required element is the file name. You can extensions to identify 'types' of files. For example, you could use QPX for your PREPARE specification files, or SPX for your Mentor specification files. Be careful not to use standard DOS or CfMC extensions to identify your files.

The spec file can include references to other files, bringing in separate portions of specifications.

If you want to enter commands line by line, press Enter at the Spec File prompt to tell the program that the input for this run will come from your keyboard.

List File

The List File is where the list of program commands processed and resulting messages go. Press Enter to tell the program that you wish to see these at your screen. Specifying PRN as the list file will send the output to the printer on most DOS machines; LP or LJ works on most MPE systems. You may also specify a file name and the output will be stored in a disk file of that name. A minus sign (-) before the name will purge a same named file. >PURGE_SAME does not affect the List File. The List File also has parameters for the page width, page length, top margin and bottom margin. See >PRINT_FILE for information on these.

Example:

```
ACME.LFL,TOP_MARGIN=5,BOTTOM_MARGIN=5
```

When you specify the list file name, if you put a plus (+) before the name then you're asking to append to the file if there is one, else to create a new one.

The spec file and list file may also be specified as part of the command line on DOS or UNIX machines:

Example:

```
PREPARE BANK.QPX -BANK.LFL (DOS, MPE)
```

Example:

```
PREPARE BANK.QPX "-BANK.LFL" (Windows NT, UNIX)
```

Because the spec file (BANK.QPX) and the list file (BANK.LFL) are specified, you would not be prompted for them. The program would process the commands from BANK.QPX and display the commands processed and resulting program messages in the file BANK.LFL. If BANK.LFL already exists, it will be purged and replaced with the new version because of the dash preceding the name.

The keyboard and console(computer screen) can also be specified as the spec and/or the list file on the command line. "CON" is the file name used for this type of standard input and output:

Example:

```
Mentor CON CON
```

If you give a file name as the spec file, Mentor expects to read that file only (and any references within that file) and then exit. If you want to read a file to start a questionnaire building session and then continue interactively from the end of that file reference on, you can use an ampersand(&) [DOS] or ampersand and the file name in quotes [Windows NT, UNIX].

Example:

```
PREPARE &BANK.SPX CON (DOS)
```

Example:

```
PREPARE "&BANK.SPX" CON (Windows NT, UNIX)
```

In these examples the file would be read and processed to the end, then the program would prompt for more commands.

Command Line Keywords

Several parameters can be specified for any CfMC program from the operating system command line. Windows NT/UNIX users, be sure to put these keywords in quotes to protect them from being interpreted by the operating system.

Example:

```
Mentor "&FILE.SPX" "-FILE.LFL" "CORE=2500000"
```

Here is a list of the command line keywords, they are explained in detail below.

CONFIG:<config file>
CORE:bytes
DEFINE:@<keyword>=<value> DEFINE:@<keywordn>=<valuen>
DUMP:switch
INFILE:<spec file>
LDEV:num (MPE and DOS only)
LISTFILE:<list file>,option1,...
SPEC_WID=###

CONFIG:<configfile>

This is used to override the default configuration file that CfMC programs must read to start up. Each release has an INITEXAM and MENTEXAM files (in the cfmc\control directory) that are examples of initial and mentinit files. They have many different options and a short explanation of them. To use these files, take the comment marks off the beginning of the line of the command(s) you want to use, and rename the file to INITIAL or MENTINIT. The command(s) will be in effect each time you use CfMC software (the initial file) or Mentor (the mentinit file). You can have more than one initial file. Here is the order in which initial files are read, by platform:

Platform	Read first	Read second	Read last
DOS	\CFMC\CONTROL\INITIAL	Not applicable	INITIAL in current dir
MPE	INITIAL.CONTROL.CFMC	INITIAL.CONTROL.account	INITIAL in current grp
UNIX	/cfmc/control/initial	$HOME/initial	initial in current dir

If you have only one initial file, it must be in the CFMC CONTROL direc-

tory/group. On UNIX and MPE, this hierarchy of initial files allows you to have an initial file that sets defaults for your site, and in addition, you then can have an initial file for personal preferences, and you can also have an initial file that applies only to a particular project.

CORE:<bytes>

Specifies the amount of memory (in bytes) that the program will use. In general, more memory allows faster processing. The keyword CORE: is required, but can occur in any position on the command line. The core value defaults to approximately 2,000,000 bytes for all programs. You can increase core, but you are limited by available system memory. To see how much core is available type the >VERSION command after loading the program. On line five the first number inside the parentheses indicates available core.

Example:

```
Prepare v.7.1 (1,29Jul97)(con,out) ... 7.2 (C) CfMC 1978 -
1995

>version

Message file: 'c:\cfmc\control\msg9705'

Date line: 9705 24july97

System versions: 725 ... 671 ... 0 ... 9601

CfMC program Prepare v.7.1 (1,29Jul97) (562838,19205)

CALLNAME="Prepare", FORCE_CON: 0, indlevel: 0, random:

12051205,0

...version info: CfMC version 7.2

...language: fr=french sp=spanish ge=german

ldev :205:
```

In this example, the PREPARE program has 562,838 bytes of core available.

DEFINE@<keyword>=<value> (=<valuen>)

Sets the value of a keyword. <keyword> can be any 1 to 16 character word that is then referenced inside a Mentor run. Keywords cannot include spaces.

Example:

```
Mentor con con DEFINE:@PATH=!CFMC!
```

This will put the environment variable or PATH into the current run.

DUMP:switch

Specifies a dump switch to be used during the current session. The switch may be any >DUMP parameter.

INFILE:<spec file>

Specifies the spec file name to be used during the current session. The keyword INFILE: is optional if the spec file name is entered as the first parameter following the program name.

LDEV:#

In MPE, this specifies the logical device number used to run a job on a terminal other than the current user's terminal.

In DOS, it specifies the communications file number for supervised Survent. Programs will open a file named @IPC#.PUB in the directory named by the environment variable CFMCCFG, where @ can be S, B, W or M. This is used by networks other than ARCNET.

UNIX: not used

The keyword LDEV: is required.

LISTFILE:<listfile>,option1 ...

This specifies the list file name and optional printing parameters to be used during the current session. The keyword LISTFILE: is optional if the list file name is entered immediately following the spec file name. The list file name can be specified as NULL or $NULL to eliminate program output. This option could be used to test program specifications.

Example:

```
Mentor myspecs lplist
```

or

```
Mentor con con
```

or

```
Mentor INFILE:myspecs LISTFILE:lplist
```

or

```
PREPARE job1.spx NULL
```

or

```
Mentor myspecs LP
```

*<filename>

In MPE, this specifies a filename that back-references a file equation. In the following example, LIST FILE output is sent to LDEV 41, but the LDEV is referenced by a name, "JANE", preceded by an asterisk (*).

Example:

```
:FILE JANE:DEV=41
.
.
:List file-->*JANE
```

List file options

The following options may be used with the list file: (see >PRINT_FILE for more information on these):

APPEND Causes the old file to be opened and added to (Default = FALSE)

BOTTOM_MARGIN=# Number of blank lines at bottom of page (Default =3)

COPIES=n In MPE, this specifies the number of times to print a spool file.

ECHO Displays lines written to a file also to the screen.

FORMS="forms message" In MPE, this specifies the forms message for spool files.

HEADER_PAGE Prints a leading header page with the following information:

Note: *Specify this option in either the INITIAL or MENTINIT file to make it the default for all sessions, as follows:*

```
>PRT; HEADER_PAGE.
```

LASER_CONTROL=<filename> Specifies the name of a control file for a laser printer. The control file must be in the current directory or group, or in the CFMC CONTROL directory or group.

This file name can have an extension, e.g.,LJTABLES.LSR. The control file contains either the initial string or specifies the name of an initial file. It can also contain the strings to control printing of standard text enhancements such bold,

underline, or color when the corresponding backslash (\) command is encountered in text. You can also specify up to 26 different user-defined strings to control special printing needs when the corresponding user-defined backslash (\~<letter>) is encountered in text.

When the program encounters a given backslash (\) option in the text it is replaced by the appropriate print string specified in this file. If there is no corresponding print string specified, then the backslash command is not passed to the print file. When an end enhancement code (e.g., \E or \-~<letter>) is seen, the OFF string for the ON item in effect at that time is used.

If no laser control file is specified, backslash (\) codes in the text are not passed to the print file.

You can have multiple laser control files that reference the same initial file or string, but supply different details for printing standard text enhancements and/or user-defined enhancements. Likewise, you could have a primary laser control file and multiple initial files to control printing in different fonts and pitch for instance.

You would have to edit the control file to specify the name of a different initial file.

Many of the keywords listed below expect a string to be specified after the equal sign (=). In general, a string must be defined inside quotes (" ") if it contains special characters or spaces. You might be able to omit the quotes, but this could cause an error from the program or unexpected results in the print file.

Note: You may specify either an INITIAL_FILE or an INITIAL_STRING, but not both in the same control file.

Control file keywords

CLOSE_STRING= Specifies the string to write out when this file is closed, e.g., to reset the printer back to its default settings.

CONTROL_SIZE=# Specifies how much room (in bytes) to allocate in core memory for the aggregate of all control strings other than the initial string. The default is 3000 and can be a minimum of 100 for smaller memory machines. Adjust this number only if you encounter memory problems.

ESCAPE_CHARACTER= Specifies the character to convert to ESCAPE. The default is asterisk (*).

EXTRA_WIDTH=# Specifies how many additional characters to allow per printed line for control codes. The default is 100. This might be needed for long print strings that could cause the text line plus the escape sequence to overflow the maximum line length.

INITIAL_FILE= Specifies the name of another file that has the initial informa-

tion in it. If the name starts with an exclamation (!) then the program will look in the CFMC CONTROL directory or group for the file named here.

INITIAL_SIZE=# Specifies how much room (in bytes) to allocate in core memory for cracking the initial string. The default for INITIAL_FILE is 300 and can be a minimum of 80 for smaller memory machines. The default for INITIAL_STRING is 1000 and can be a minimum of 300. Adjust these numbers only if you encounter memory problems.

INITIAL_STRING= {<escape sequences>} Specifies the initial control string. This string is defined within left and right braces { } and may continue to more than one line.

Note: For HP Laser printer escape sequences, break lines so the each new line begins with an escape character; otherwise, the printer will not know that it is supposed to do something special with the characters and will simply print them.

NEW_PAGE= Specifies what to do for a new page instead of <Ctrl>-L.

NEW_LINE= Specifies what to do for a new line.

PAGE_LENGTH=# Specifies page length if it has not been specified on either >PRINT_FILE or LISTFILE. Page length specified on either >PRINT_FILE or LISTFILE always overrides this value.

PAGE_WIDTH=# Specifies page width if it has not been specified on either >PRINT_FILE or LISTFILE. Page width specified on either >PRINT_FILE or LISTFILE always overrides this value.

SPECIAL_BACK_SLASH Allows you to put out a backslash (\) character for Microsoft Rich Text Format. The default is off, i.e., SPECIAL_BACK_SLASH.

The following keywords affect how the standard text enhancements \B, \F, \I, \U, and \W will be converted. If you specify an ON control string then you must specify the corresponding OFF control string.

BOLD_ON= Specifies the string to use for \B.

BOLD_OFF= Turns \B off when either an end bold (\-B) or an end enhancement (\E) is encountered.

FLASHING_ON= Specifies the string to use for \F.

FLASHING_OFF= Turns \F off when either an end flashing (\-F) or an end enhancement (\E) is encountered.

INVERSE_ON= Specifies the string to use for \I.

INVERSE_OFF= Turns \I off when either an end inverse (\-I) or an end enhancement (\E) is encountered.

UNDERLINE_ON= Specifies the string to use for \U.

UNDERLINE_OFF= Turns \U off when either an end underline (\-U) or an end

enhancement (\E) is encountered.

WIDE_ON= Specifies the string to use for \W.

WIDE_OFF= Turns \W off when either an end wide (\-W) or an end enhancement (\E) is encountered.

Note: *These text enhancements are automatically turned off at the point where a stub label would wrap, and turned back on for continuation stub label lines. Only the stub label is enhanced and not the entire row. Likewise, these enhancements are turned off either at the end of a banner or a title. For banners and titles, this means you do not have to specifically supply a \E in the text except where you specifically want to turn an enhancement off.*

The following keywords control color enhancements. \C, \D, and \DD are all controlled with these keywords. You must specify control strings for all the color keywords. A \E in the text ends the current color enhancement and resets it to the default.

- The following turn off foreground color and returns to the default setting.

```
COLOR_FOREGROUND_BLACK=
COLOR_FOREGROUND_BLUE=
COLOR_FOREGROUND_GREEN=
COLOR_FOREGROUND_CYAN=
COLOR_FOREGROUND_RED=
COLOR_FOREGROUND_MAGENTA=
COLOR_FOREGROUND_YELLOW=
COLOR_FOREGROUND_WHITE=
COLOR_FOREGROUND_DEFAULT=
```

- The following turn off background color and returns to the default setting.

```
COLOR_BACKGROUND_BLACK=

COLOR_BACKGROUND_BLUE=

COLOR_BACKGROUND_GREEN=

COLOR_BACKGROUND_CYAN=

COLOR_BACKGROUND_RED=

COLOR_BACKGROUND_MAGENTA=

COLOR_BACKGROUND_YELLOW=

COLOR_BACKGROUND_WHITE=

COLOR_BACKGROUND_DEFAULT=
```

The following keywords control up to 26 (A-Z, case-insensitive), different user-defined enhancements. The appropriate ON control string is substituted for each \~<letter> encountered in the text; \-~<letter> ends the user-defined enhancement and the corresponding OFF control string is used. You must supply both an ON and an OFF string for every \~<letter> and \~-<letter> specified in the text. User-defined enhancements are never turned off by the program until the corresponding \~-<letter> is seen.

USER_<A-Z>_ON= Specifies the string for \~<A-Z>.

USER_<A-Z>_OFF= Specifies the string for \~-<A-Z>.

Example laser control file for a HP LaserJet printer:

```
ESCAPE_CHARACTER=*

''Print Tables

INITIAL_STRING={

^[E^[&l1o66p5e1L^[(8U^[(s0p16.66h0s-
3b0T^[&a8L^[&l5.4545C^[&k7H }

BOLD_ON="*(s3B"

BOLD_OFF="*(s0B"

CLOSE_STRING="*E"
```

What follows is the laser control file needed to create Microsoft Rich Text Format (RTF) files. The enhanced text file will contain embedded escape sequences (specified here) to control text enhancements, fonts, etc., and will main tain these when

the file is imported into word processing software.

```
PAGE_LENGTH= 59
PAGE_WIDTH= 132
EXTRA_WIDTH= 100
SPECIAL_BACK_SLASH
INITIAL_SIZE= 400
INITIAL_FILE= !lasrinit
'INITIAL_STRING= {
'This is where the initial string
'would be specified. It might
'contain multiple lines.
'}
CONTROL_SIZE= 3000
''wide used to control shadow
WIDE_ON = "\shad "
WIDE_OFF = "\shad0 "
UNDERLINE_ON = "\ul "
UNDERLINE_OFF = "\ulnone "
''inverse used to control outline
INVERSE_ON = "\outl "
INVERSE_OFF = "\outl0 "
''flashing used to control italic
FLASHING_ON = "\i "
FLASHING_OFF = "\i0 "
BOLD_ON = "\b "
BOLD_OFF = "\b0 "
COLOR_FOREGROUND_BLACK = "\cf0 "
COLOR_FOREGROUND_BLUE = "\cf1 "
COLOR_FOREGROUND_GREEN = "\cf3 "
COLOR_FOREGROUND_CYAN = "\cf2 "
COLOR_FOREGROUND_RED = "\cf5 "
```

```
COLOR_FOREGROUND_MAGENTA = "\cf4 "
COLOR_FOREGROUND_YELLOW = "\cf6 "
COLOR_FOREGROUND_WHITE = "\cf7 "
COLOR_FOREGROUND_DEFAULT = "\cf0 "
"setting background color does nothing in RTF files
COLOR_BACKGROUND_BLACK = ""
COLOR_BACKGROUND_BLUE = "\cb1 "
COLOR_BACKGROUND_GREEN = "\cb3 "
COLOR_BACKGROUND_CYAN = "\cb2 "
COLOR_BACKGROUND_RED = "\cb5 "
COLOR_BACKGROUND_MAGENTA = "\cb4 "
COLOR_BACKGROUND_YELLOW = "\cb6 "
COLOR_BACKGROUND_WHITE = "\cb7 "
COLOR_BACKGROUND_DEFAULT = "\cb0 "
NEW_PAGE = "\page "
NEW_LINE = "\par "
CLOSE_STRING = "}"
"USER_A_ON = "USER-A-ON"
"USER_A_OFF= "USER-A-OFF"
```

LASER_NUMBER= # Specifies which laser printer to use (LJET1, LJET2, etc.). This option is only applicable to the MPE operating system. Your laser printers must be configured as LJET1, LJET2, etc., and the number (#) specified here will be appended to “LJET” to indicate the device to use.

LP In MPE, this specifies the line printer for output from a run instead of a file.

Example:

```
List file-->LP
```

PAGE_LENGTH= # Sets the total number of print lines on a page (Default =66)

PAGE_WIDTH= # Sets the number of characters per print line (Default =132)

TOP_MARGIN= # Sets the number of blank lines at top of page (Default = 3)

Example:

```
Mentor myspx mylist,ECHO,LASERC=ljtables,LASERN=2
```

In MPE, if the “Listfile” is “LP”, then LASER_NUMBER= redirects the output to the device called LJET1, LJET2, etc.

SPEC_WID:###

Sets the spec file width for up to 5000 characters. The default file width is 5000.

Example:

```
Mentor "file.spx" "-file.lst" "spec_wid:500"
```

Using DOS Variables in the File Name

You can set a variable at the DOS level i.e., ‘SET DRIVE=C:’, and reference it wherever you can reference a file name in CfMC programs. This would be at the Spec or List File prompts, or when using &filename references. To reference the variable ‘DRIVE’, put exclamation points (!) around it at the start of the file name reference, e.g., !DRIVE!Myfile.LST would look for C:Myfile.LST.

The variable named ‘CFMC’ has a special meaning. Do not use it for other uses.

Wherever you set it is where the program(s) look for CFMC files. The default for the variable CFMC is “\CFMC\”.

Program-Generated File Extensions

Mentor, PREPARE, and the utility programs produce some files automatically, and some when you request them. Here the files thatCfMC software can create if your study name is DEMO:

Extension	File type	File name: DOS UNIX	MPE:
ASC	ASCII	DEMO.ASC	DEMOASC
BIN	Binary output	DEMO.BIN	DEMOBIN
CHK	Data list	DEMO.CHK	DEMOCHK
CLN	Cleaning specs	DEMO.CLN	DEMOCLN
CNT	FREQ raw count	DEMO.CNT	DEMOCNT

Extension	File type	File name: DOS UNIX	MPE:
CSP	C specs from SPC	DEMO.CSP	DEMOCSP
DB	database	DEMO.DB	DEMODB
DEF	Tab set definitions	DEMO.DEF	DEMODEF
DLM	Delimited output	DEMO.DLM	DEMODLM
DTA	SPL data file	DEMO.DTA	DEMODTA
FNX	Phone index 1	DEMO.FNX	DEMOFNX
FNY	Phone index 2	DEMO.FNY	DEMOFNY
FNZ	Phone index 3	DEMO.FNZ	DEMOFNZ
FON	Survent phone file	DEMO.FON	DEMOFON
FRQ	FREQ print file	DEMO.FRQ	DEMOFRQ
HRD	Hardcopy	DEMO.HRD	DEMOHRD
HOL	Holecount	DEMO.HOL	DEMOHOL
HPS	SPL Survent specs	DEMO.HPS	DEMOHPS
LAB	SPL Labels	DEMO.LAB	DEMOLAB
LPR	Print and load	DEMO.LPR	DEMOLPR
LST	List output	DEMO.LST	DEMOLST
PRT	Print file	DEMO.PRT	DEMOPRT
QSP	Survent specs in ASCII	DEMO.QSP	DEMOQSP
QFF	Survent questionnaire	DEMO.QFF	DEMOQFF
QUO	Quota	DEMO.QUO	DEMOQUO
RCD	RECODE changes list	DEMO.RCD	DEMORCD
RFL	REFORMAT data locks	DEMO.RFL	DEMORFL
RFT	REFORMAT output	DEMO.RFT	DEMORFT

Extension	File type	File name: DOS UNIX	MPE:
SCN	Scan output	DEMO.SCN	DEMOSCN
SUM	Question list	DEMO.SUM	DEMOSUM
SWB	Swap binary file	DEMO.SWB	DEMOSWB
TAB	Table specs	DEMO.TAB	DEMOTAB
TR	Data file	DEMO.TR	DEMOTR

Ampersand Referencing (&<specfile>)

Sometimes you may want to keep parts of a specification separate (keep a demographic section of a questionnaire separate, for instance, so that you can call it into several jobs), but need to call in the pieces when you do the complete run.

You can do this by using the ampersand (&). When working interactively (having pressed Enter to the SPEC FILE prompt, or having specified CON on the program line), you call in a file by entering the ampersand followed by the file name.

Example:

```
-->&demos
```

This example will call in the file DEMOS at that point in the program run. The ampersand reference can include a path reference if necessary.

When accessing files with the use of the ampersand “&”, it is possible to control where the statements being accessed will print. This is accomplished, optionally, by the use of a plus “+” or minus “-” sign after the ampersand (before the filename). The plus sign lists the file both at the location requested at the LIST FILE prompt and on the terminal in use (“+” signifying more than one location); the output from the run, however, will print where you've specified on your LIST FILE prompt. The minus sign tells the program not to list these lines anywhere (“-” signifying less than one location).

If the file you are accessing has an extension (DOS/UNIX only), then you may want to use a caret (^) in place of the standard dot (.) before the extension. For example, demo.DEF becomes demo^DEF. In this way, the file referenced can be read in by any CfMC-supported operating system. Using our example, the MPE operating system would correctly understand the file name as demoDEF and not demo.DEF.

Use the meta command >ALLOW_INDENT if you prefer to indent &specfile ref-

erences in the specifications file or Mentor procedure from which it will be called.

Ctrl-Y or Ctrl-C or Ctrl-<intr>

Ctrl-Y shows you current information on a program run and prompts you as to whether you want to break out of programs. This guarantees that all files are closed properly on any system, and all work done to that point is saved.

You can modify the interrupt key in unix using the 'stty intr' command. In DOS the interrupt key is Ctrl-C.

Ctrl-Y behaves differently depending on which CfMC program you are running, and what the program is doing at the point Ctrl-Y is issued.

In Mentor, Ctrl-Y works as follows:

- If the program is reading a data file then Ctrl-Y will print the record number of the current case and its ID. Setting >CONTROL_Y_QUIET would allow you to check the current case without having to respond to the terminate confirmation.
- If the program is defining variables (~DEFINE) then Ctrl-Y will print the name of the last variable defined.
- If you are reading cases interactively (~CLEANER NEXT, FIND, HUNT, or FIND_FLAG) then Ctrl-Y will print "you stopped case reading so no case in core" and resets the input file to the beginning of the file.

In PREPARE, Ctrl-Y will terminate the compile (after getting confirmation) and exit to the operating system.

For the phone system, Ctrl-Y works as follows:

- For SURVSUPR or STDYSRVR it will give you a prompt so that a command can be entered.
- For FONEUTIL it will print the current record number of the total number of records and then prompt for confirmation before terminating the process.

Renaming Files ($filename)

Mentor users can specify their own file names for files that normally expect a standard CfMC file extension (such as TR, DB, or PRT) by preceding the file name with a dollar sign ($). For example, ~INPUT $JAN.DAT. Length of file-

name is limited to the maximum allowed by your operating system: eight plus a three-character extension for DOS; eight for MPE; and 14 for UNIX, although some newer versions have no character limit.

Where file type can be checked (currently only on MPE) the program will check and complain if there is a conflict.

The meta command >-CFMC_FILE_EXTENSIONS makes $filename the default.s

Variables

Variables are used extensively in Mentor, but are also used in the Utilities for bases, weighting, etc. There are two basic variable types, the punch variable and the field variable.

Punch (caret) Variables

The caret (^) can be used to create a punch variable that goes across one or more columns.

Using the following chart we will show how this works without having to enter each column and its punches individually.

A ^ variable has a maximum of 21 columns of data. It basically converts the punch to a sequence number, so that the punches in the first column specified are 1-12, in the second column they are 13-24, in the third 25-36, etc.

The numbers must be separated with commas for separate items, or a period or dash for a range (1,2,3 means 1 or 2 or 3; 1-5 means 1 or 2 or 3 or 4 or 5).

PUNCH 1 2 3 4 5 6 7 8 9 0 X Y

Punch column	1	2	3	4	5	6	7	8	9	0	X	Y
1	1	2	3	4	5	6	7	8	9	10	11	12
2	13	14	15	16	17	18	19	20	21	22	23	24
3	25	26	27	28	29	30	31	32	33	34	35	36

Example: -->[2/5.3^01-10,23,30]

Field Variables

Field variables are actual numbers signifying real amounts, or values which make up a character string. When using a field variable, a pound sign (#) separates the column location and value specifications (replacing the caret of a punch variable). Sequential values are specified by placing a dash (-) between the first and last of the series, and commas (,) separate non-sequential values.

Example:

```
-->[4/10.5#15000-60000]
```

If the characters specified look like numbers, they will be treated as such. If you are looking for an exact string of numbers (i.e., 001 not the numeric value of 1), enclose the characters in quotation marks (`[3/4.3#"001"]`). If the character string you're defining is all alphanumeric characters, quotation marks are unnecessary; other characters must be enclosed in quotation marks (i.e., `[3/30.3#ABC,DEF,"S,Z"]`). Numeric values and literals can both be in the same definition.

Example:

```
-->[4/17.3#1-10,REF]
```

If you define a field larger than the number of characters specified, it's important that you understand how the program looks in that data location for the match.

- Numbers must be right-justified in the field with optional leading zeroes. Leading blanks and zeroes are treated the same as long as once a numeric character is seen, all remaining columns are also numeric. A plus (default) or minus sign may immediately precede the first numeric character.

Numbers	Not numbers
001	- 5
7	0 3
-5	7

- Character strings are searched for by starting in the left-most column of the field. If you are looking for characters (whether or not they look like numbers), you should enclose them in quotations for an exact match, otherwise know that they must be left-justified in the field, must be the only characters in the field ([2/10.4#dk] will match "dk" or "dk " but not "dkna" or " dk"), and any characters that are not alphanumerics *must* be enclosed in quotation marks. The search for alpha characters is *not* case-sensitive.

Complex connectors (for bases, etc.)

It is possible to select cases for analysis which have met the conditions of two or

more specified criteria. This is done by entering separate variable specifications in any of the manners explained above. To enter a complex statement, enter a left bracket "[", the first variable specifications and a right bracket "]". This is followed by a space then one of four connectors:

1. **"AND" requires that both conditions are met.**
2. **"OR" requires that at least one or the other condition is met.**
3. **"[connector] NOT" varies depending on the connector (AND, OR) used.**
4. **"XOR" (for "EXCLUSIVE OR") requiring that one or the other condition is met, but both are not allowed.**

After typing in the connector, enter a space, then type another left bracket, another variable specification statement and a right bracket. If you want to use more than two connectors, enter the information within brackets for each new specification and add the proper connectors. Use parentheses to control the order in which the specifications are checked if you use different connectors in the same statement.

Example:

```
-->[1/5^1-3] AND [3/60.2#CA,NV,AZ,NM]
```

This example will select only those cases having both a 1, 2, or 3 punch in column 5 AND the letters CA, NV, AZ, or NM in columns 60-61 of record 3. If cases do not meet both criteria, they are not included.

Mentor users should refer to the *Mentor* manual for more connectors.

Other Useful Commands

>QUIT

Most CfMC programs will automatically abort at any desired point when `>QUIT is entered` after an arrow prompt. This returns you to the operating system. Use this with care because you will lose changes you have made to any data files that are currently open.

Example:

```
-->>QUIT
```

>DUMP I7

>DUMP 17 saves spec files generated by the utilities. This is useful when you

want a person unfamiliar with the CfMC software to repeatedly (on a daily basis, for example) run a utility program without having to answer all the prompts. You can run the program once yourself, using >DUMP I7 to save your responses, then have the other person run Mentor using your file as the spec file.

>SYSTEM command

>SYSTEM allows you to access information available from normal operating system commands without breaking a CfMC program run (see *Appendix A: META COMMANDS* for more information on this command).

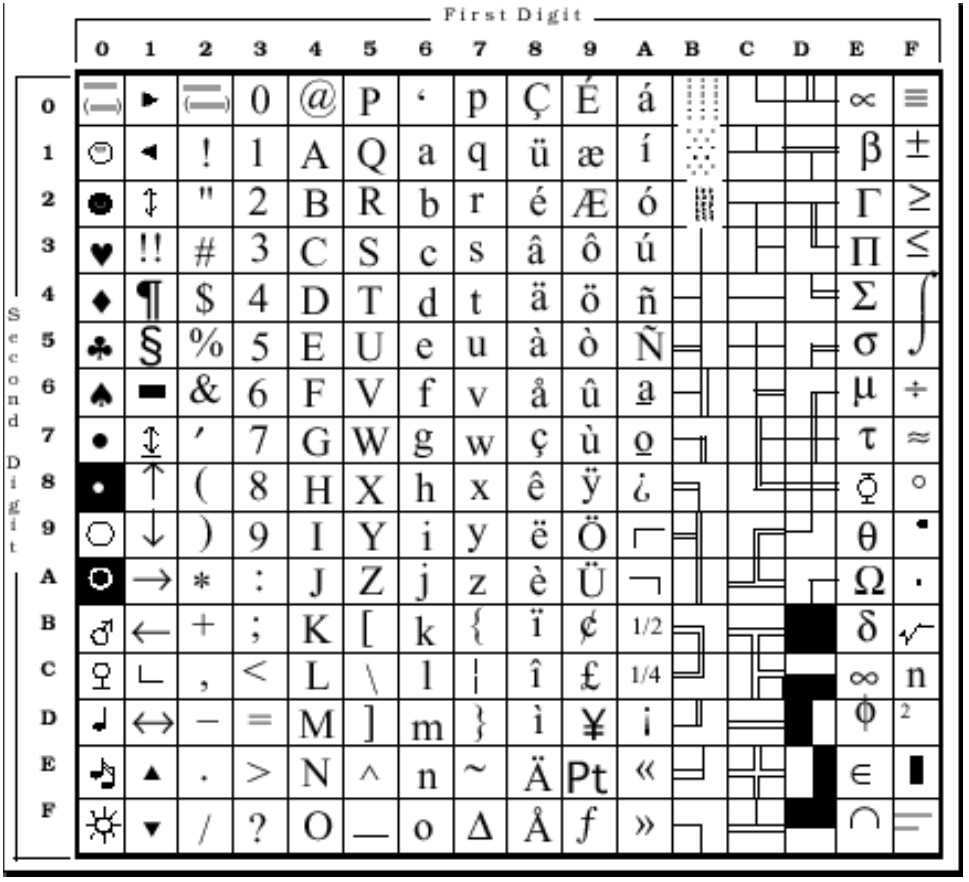
>PUT_CHARACTERS

>PUT_CHARACTERS allows you to specify characters that designate how to fill a field containing zeroes (either integer or floating point) or missing data (see *Appendix A: META COMMANDS* for more information on this command).

>LIST_DB_CONTENTS

This command is used to generate a list of items stored in a DB file (see *Appendix A: META COMMANDS* for more information on this command).

Appendix E
Graphic Characters



Hexadecimal Values may be used to display a variety of special characters and symbols in question text. See 2.5.2 TEXT ENHANCEMENTS AND

GRAPHIC CHARACTERS in the Survent manual or Appendix B: Tilde Commands ~CLEANER PRINT_LINES in the Mentor manual for information on how to use these graphics.

Enter \^and the two-digit hexadecimal value corresponding to the desired symbol. The first number represents the horizontal value. The second number represents the vertical value.

In this example, the text is followed by a smiley face (hexadecimal value 01):

```
THAT CONCLUDES OUR INTERVIEW. THANK YOU \^01
```

NOTE: *CfMC recommends that you use your browser to search for information regarding Graphic Characters for your particular Operating System.*

Appendix F
CON and ZSPC Statements

CON Statements

CON statements are Survent commands used to control the passing of statements to Mentor or other programs to be processed. In this way, you can write utilities that request input in a Survent-like application, and then execute commands to do tables, write files, etc. This is how the CfMC utility set is developed.

Syntax:

{<optional label>
!IF <optional condition controlling command>
.<lines to be sent to program>
!CON,<type>,<options>
}

In addition to sending commands, !CON statements can control what happens after they are executed (whether it stays in Survent, it exits Survent and comes back to the beginning, it exits Survent and quits, it exits Survent and returns to the current position in the questionnaire).

!CON,1

Sends textlines to be processed at next !CON 2-4 or end of program, stays in the program.

Example:

{filename:
Enter the name of your input file:
!var,,70}
{!if filename$<>" "

```
.>purgesame
.~input \:filename:
!con,1}
```

You use Survent to ask the questions you need to fill in your program, such as the name of the file to use, and send the !CON statement to the controlling program.

Notice the period at the front of the text line on the !CON,1 statement. That character is a “holding” character and will not be passed to mentor. It allows you to have meta commands be sent without them being executed when you compile the script.

You can save the lines to send into a file in case you want to process them later instead of immediately when exiting. Here is a list of the !CON types:

!CON,1,<name>Saves a file of the lines to execute to directory "name" with name 'mnt<yyyymmddhhmmss>.cmd'

!CON,1,file=name Saves file of lines to execute to local file "name".

!CON,1,file="\:label:" Saves file of lines to execute to local file from the question label "name". This way, you can ask for a particular name to save with.

!CON,1,file=[location]Saves file of lines to execute to local file from the data location "location".

!CON,2 Executes lines from !CON,1 statements and exits

!CON,3 Executes lines from !CON,1 then returns to the top of the questionnaire

!CON,4 Executes lines from !CON,1 then returns to current positon in the questionnaire

To run questionnaires using !CON statements to control CfMC programs, you must call the compiled questionnaire script from the “Spec File” prompt using a “&&&<filename>” reference, for example:

```
mentor “&&&${CFMC}support/scan.qff” con
```

This command line will run the “scan” utility in Mentor. The “con” in the list file position says that input is coming from the “console” which is the case when running a Survent utility.

Currently you can use !CON statements in Mentor or Survsupr.

ZSPC Statements

ZSPC statements are used in writing Mentor or Survent utilities or applications to get more information on data files, DB items, or other system uses, such as date/time formatting. Both Survent and Mentor use ZSPCs to get immediate access to system information.

Syntax (Survent):

```
!ZSPC,type,subtype,[col.1],[col.wid1],[col.wid2],...,[col.widn]
```

Syntax (Mentor):

```
ZSPC,type,subtype,[col.1],[col.wid1],[col.wid2],...,[col.widn]
```

[col.1] is the one column wide field to hold the return code (the 1 is optional)
[col.wid1] is the field holding (or to hold) the information for Argument 1
[col.wid2] is the field holding (or to hold) the information for Argument 2, etc.

For all types, a return code of D means that there is no data case to deal with. A return code of Z means there were zero arguments.

ZSPC TYPE 1

This type is used for date/time format conversions.

TYPE 1, subtype 1

Converts an ASCII date to a standard date format.

ASCII date format: `07dec97 19:03`
Standard date format: `97120719037341`
YYMMDDHHMMWJJJ

- Year, month, day, hour, minute, day of week, and Julian date)
- The day of the week is a number from 1 (Monday) to 7 (Sunday).
- The Julian date is the number of days since the start of the year.

Syntax:

```
ZSPC,1,1,[return code],[arg1],[arg2]
```

Argument 1 is ASCII date to convert. Multiple date formats are acceptable (12/7/97, 7dec, 09:00, 9:00). ZSPC 1 will accept the same date and time formats that you can use with the Phone System's WHEN TO CALL BACK prompt (See

Survent Manual, Chapter 6).

Argument 2 is location to return the converted standard date format; this must be 6-14 positions wide.

Return codes:

A	Not two arguments
1	Bad ASCII date in Argument 1
2	Bad length in Argument 2; should be 6-14
0	All OK

Type1, subtype 2

This is the reverse of subtype 1.

Syntax:

```
ZSPC,1,2,[return code],[arg1],[arg2]
```

Argument 1 is standard date format (see above).

Argument 2 is location to return the converted ASCII date format; this must be 15-22 positions wide.

Return codes:

A	Not two arguments
1	Bad standard date in Argument 1
2	Bad length in Argument 2; should be 15-22
0	All OK

TYPE 1, subtype 3

Adds years, days, hours, etc., to the standard date format and returns the new date in standard format.

Syntax:

```
ZSPC,1,3,[return code],[arg1],[arg2],[arg3]
```

Argument 1 is date/time in standard format.

Argument 2 is the same format, with only the items you want to add.

Argument 3 is location to store the new date/time in standard format; this must have a length from 6 to 14.

Example: Argument 2

YY	MM	DD	HHMMW	JJJ	
01					adds one year
	06				adds six months

	14			adds 14 Days
		0318		adds 3 hours, 18 minutes
	250			adds 250 days
03		1520	100	adds 3 years, 15 hours, 20 minutes and 100 days

Special conditions on use:

- You can change only one of MMDD or JJJ at a time.
- The Julian date (JJJ) cannot be greater than 255.

Note: When MM is used, the program keeps the day as the last day of the month if it starts out that way.

Return codes:

A	Does not have three arguments
1	Bad standard date in Argument 1
2	Bad standard date in Argument 2
3	Bad length in Argument 3, must be 6-14
0	All OK

ZSPC TYPE 2

This type returns information on data files and other system elements.

TYPE 2, subtype 1

Opens a CfMC TR file and checks whether a particular data case exists in it.

Syntax:

```
ZSPC,2,1,[return code],[arg1],[arg2]
```

Argument 1 is the case ID you want to check.

Argument 2 is optional and, if specified, indicates the name of the TR file to be checked. If argument 2 is not specified then default to the input TR file.

Return codes:

A	More than two arguments
1	Case ID length in Argument 1 is more than 10
2	Argument 2 has bad data file name
O	Cannot open TR file named
I	No input data file and no Argument 2
Y	Case found

N	Case not found
F	ASCII input data file
B	BINARY input data file
D	DTA type input data file
S	SWAP-BINARY input data file
P	Phantom file, no case ID
J	Joined files, cannot use

TYPE 2, subtype 2

Checks DB items to see if they exist, and what type of item they are.

Syntax:

```
ZSPC,2,2,[return code],[arg1],[arg2]
```

Argument 1 is the name of the DB item to check; it can be 8 or 16 wide.

Argument 2 if specified, is where information about the DB item is to be returned. The default is do not return.

Argument 3 if specified, is the name of the DB file to look in. Default: any open DB file. May be 'LOCAL' for the local DB file.

Data returned if argument 2 is specified:

type	6 bytes	Variable, table, etc.
gloss1	6 bytes	If variable, is type of variable (0-9)
gloss2	6 bytes	More info about the item
instance	6 bytes	Nth occurrence of this item (if multiple)
entry num	6 bytes	This is the directory slot used for this item
byte offset	10 bytes	This is the file position of this item
byte count	6 bytes	This is the byte length of item as stored
when put	14 bytes	This is standard time format when item stored: YYMMDDHHMMWJJJ

Return codes:

A	More than 3 arguments
1	Length in Argument 1 more than 16
2	Length in Argument 2 bad (should be 6-60)
3	Argument 3 is bad file name
F	DB file not defined
N	Item not in DB file
Y	Item is in DB file

TYPE 2, subtype 3

Returns the last error code seen (i.e. the number you see on error messages such as:

```
(ERROR #2640) expected ~INPUT keyword
(ASCII=#, STOP_AFTER=#, SELECT=, etc.)
```

In this case, the ZSPC,2,3 would return the number “2640”.

Syntax:

```
ZSPC,2,3,[return code],[arg1.5]
```

Argument 1 is where to return the error code, and must have a width of 5.

Return codes:

A	Not 1 argument
1	Width not 5
0	Value of c->errors is in location

TYPE 2, subtype 4

Returns the name of the next temporary file to be made by the program.

Syntax:

```
ZSPC,2,4,[return code],[arg1.8]
```

Temporary file names have the syntax 'TE######' where ###### are sequential numbers from 1 to n for each file made.

Argument 1 is where the next temporary name will be returned with a width of 8.

Return codes:

A	Not 1 argument
1	Width not 8
0	Temporary name is in the location

TYPE 2, subtype 5

Returns whether a file exists, and if it does, it’s fully qualified (i.e., 'directory/filename' in UNIX, 'drive:directory\filename' in DOS, and 'filename.group.account' in MPE) file name.

It interprets most valid CfMC filename specifications; an @name is treated as a DEFined value, a !name! is treated as a variable set at the operating system level. Caret (^) is treated as a dot (.) in UNIX or DOS, and as nothing in MPE. Wildcard characters (*,?,#) for multiple file references are not interpreted, since the command only deals with one file at a time.

Syntax:

```
ZSPC,2,5,[return code],[arg1],[arg2]
```

Argument 1 is the name of the file to return information on and can include the CfMC characters described above.

You may include directory/group information, or if not, the program assumes the file is in the current work area.

Argument 2 Returns the fully qualified filename if used (not required).

Return codes:

A	More than two arguments
N	File not found
Y	File found

TYPE 2, subtype 6

Opens a CfMC TR file and returns detailed information about it.

Syntax:

```
ZSPC,2,6,[return code],[arg1],[arg2]
```

Argument 1 is the location for return information. Length controls how much information to get. Can be from 5 to 50 long, depending on information wanted.

Argument 2 is the name of the TR file to get. The TR part of the name is not required. It defaults to the currently open TR file.

Return codes:

A	More than 2 arguments
Y	Got file
O	Can't open file
I	No input file opened
1	Argument 1 not 5 to 50 columns
2	Argument 2 had a bad file name
D	DTA type input data file
F	ASCII input data file
B	BINARY input data file
S	SWAP-BINARY input data file
P	PHANTOM file

Return information

Item/Order to get	Data Type	Length	Position
Case length	#	5	1-5
Number of cases	#	7	6-12
Number of deleted cases	#	7	13-19
TR file type (0=standard, 1= indexed)	#	1	20
Maximum number of cases (approx.) only if indexed	#	7	21-27
Duplicates only if indexed Y=Yes, N=No	A	1	28
Data file comment	A	22	29-50

TYPE 2, subtype 7

Returns the amount of available core memory space.

Syntax:

```
ZSPC,2,7,[return code],[arg1.5]
```

Argument 1 location for amount of available core. Width must be five.

TYPE 2, subtype 8 Returns the tilde (~) block of Mentor that the user is in.

Syntax:

```
ZSPC,2,6,[return code],[arg1.2]
```

Argument 1 is the number of the tilde block currently in:

00	- Not in any block	01	- DEFINE
02	- RESET	03	- SORT
04	- STOP_WATCH	05	- PREPARE
06	- SET	07	- INPUT
08	- CLEANER	09	- RESTORE
10	- MAKE_ASQ	11	- QSP_FILE

12	- QFF_FILE	13	- ADJUST
14	- unused	15	- JCD (debugging)
16	- PRACTICE	17	- unused
18	- OUTPUT	19	- unused
20	- TRANSLATE	21	- COPY
22	- unused	23	- NEXT
24	- EXECUTE	25	- unused
26	- VIEW	27	- CLOSE_STUFF
28	- INTERVIEW	29	- SHOW
30	- unused	31	- FREQUENCY
32	- DROP	33	- COMMENT
34	- unused	35	- unused
36	- CY (debugging)	37	- unused
38	- NEW	39	- WRITE_SPECS
40	- UPDATE	41	- SPEC_RULES

Return codes:

A	more than 1 argument
1	Argument 1 not 2 columns wide
Y	all OK

TYPE 2, subtype 9

Returns the information off the &&&qfilename command line. This lets you say something like '&&&HELP INPUT' and have the help system start out at the INPUT block.

Syntax:

```
ZSPC,2,6,[return code],[arg1]
```

Argument 1 is the text that follows the &&&qfilename. The leading spaces are taken out of the returned text.

Return codes:

A	more than 1 argument
1	Argument 1 not long enough to hold text
Y	all OK
N	not in &&& command

TYPE 2, subtype 10

Checks printer ports in DOS or UNIX.

There are no arguments.

Return codes:

A	more than 0 arguments
Y	Printer found online
O	Printer found offline
P	Printer found out of paper
N	Printer not found

ZSPC TYPE 3

Stores or retrieves items in a CfMC DB file.

TYPE 3, subtype 1

Stores ASCII data from a file into a CfMC DB file.

Syntax:

```
ZSPC,3,1,[return code],[arg1],[arg2],[arg3]
```

Argument 1 is the name of the DB item to store (the maximum number of characters in a DB item name is 24).

Argument 2 is the name of the file to store or the location in the currently open data file.

Argument 3 is optional and is the name of the DB file to store to; the default is the current DB file opened READ_WRITE. You may say “LOCAL” for local DB file.

Return codes:

A	Not 2 or 3 arguments
1	Length of argument is one more than 16
3	Argument 3 is bad file name
E	Error in DBstore routine
0	All OK

TYPE 3, subtype 2

Stores binary data from a data file into a CfMC DB file.

Syntax:

```
ZSPC,3,2,[return code],[arg1],[arg2],[arg3]
```

Argument 1 is the name of the DB item to store (the maximum number of characters in a DB item name is 24).

Argument 2 is the name of the data file to store or the location in the currently

open data file.

Argument 3 is optional and is the name of the DB file to store to;

the default is the current DB file opened READ_WRITE. You may say 'LOCAL' for the local DB file.

Return codes:

A	Not 2 or 3 arguments
1	Length of argument is one more than 16
3	Argument 3 is bad file name
E	Error in DBstore routine
0	All OK

TYPE 3, subtype 3

Takes an ASCII DB item and writes it to a new data file.

Syntax:

```
ZSPC,3,3,[return code],[arg1],[arg2],[arg3]
```

Argument 1 is the name of the DB item to get (the maximum number of characters in a DB item name is 24).

Argument 2 is the name of the file to write out or the location in the currently open data file.

Argument 3 is optional and is the name of the DB file to use; the default is any opened DB file. You may say “LOCAL” for local DB file.

Return codes:

A	Not 2 or 3 arguments
1	Length of argument is one more than 16
3	Argument 3 is bad file name
E	Error in DBfetch routine
0	All OK

TYPE 3, subtype 4

Takes a binary DB item and writes it to a new data file.

Syntax:

```
ZSPC,3,4,[return code],[arg1],[arg2],[arg3]
```

Argument 1 is the name of the DB item to get (the maximum length of a DB item name is 24 characters).

Argument 2 is the name of the file to write out or the location in the currently open data file.

Argument 3 is optional and is the name of the DB file to use; the default is any opened DB file. You may say "LOCAL" for local DB file.

NOTE: You still cannot have variable names more than 14 characters. The only ones that can have these long names are ZSPC,3-created items.

Return codes:

A	Not 2 or 3 arguments
1	Length of argument is one more than 16
3	Argument 3 is bad file name
E	Error in DBfetch routine
0	All OK

ZSPC TYPE 4

Returns the elements of a built table.

TYPE 4, subtype 1

Returns the starred (*) table elements below and all others only if the user specified them. (Subtype 2 returns all of them always).

Syntax:

```
ZSPC,4,1,[return code],[arg1.208]
```

Argument 1 returns the table elements; it must have a width of 208. The table elements returned are (in order):

TABLE name, *HEADER, *FOOTER, *TITLE, COLUMN, ROW,*BANNER, *STUB, WEIGHT, *EDIT, LOCAL_EDIT, BASE and STATISTICS variable.

Each of the 13 elements takes up 16 characters, for a total of 208.

Return codes:

A	Not 1 argument
1	Width not 208
0	OK, information returned

TYPE 4, subtype 2

Returns all the table elements below.

Syntax:

```
ZSPC,4,2,[return code],[arg1.208]
```

Argument 1 returns the table elements; it must have a width of 208. The table ele-

ments returned are (in order):

TABLE name, HEADER, FOOTER, TITLE, COLUMN, ROW, BANNER, STUB, WEIGHT, EDIT, LOCAL_EDIT, BASE, and STATISTICS variable.

Each of the 13 elements takes up 16 characters, for a total of 208.

Return codes:

A	Not 1 argument
1	Width not 208
0	OK, information returned

ZSPC TYPE 5

Checks for information related to a DB item:

TYPE 5, subtype 1

Checks on items with an 8 wide DB name.

Syntax:

```
ZSPC,5,1,[return code],[arg1.8]
```

TYPE 5, subtype 2

Checks on items with a 16 wide DB name.

Syntax:

```
ZSPC,5,2,[return code],[arg1.16]
```

Argument 1 is the name of the DB item to check.

Return codes:

A	Not 1 argument
W	Width not 8 or 16 columns
0	Doesn't exist
1	Number (NUMeric question)
2	Categories (CATegory or FieLD question)
3	Vector (or multi-column number)
4	String (VARiable question)
5	Table
6	DB spec file
7	Exists, but is something else altogether VirusScanv

ZSPC TYPE 7

Sends commands immediately.

TYPE 7, subtype 1

Sends commands immediately (such as META commands).

Syntax:

```
ZSPC,7,1,[return code],[location to send]
```

ZSPC TYPE 11

Reads or writes a line of an ASCII file:

TYPE 11, subtype 1

Reads a line of data from an ASCII file.

Syntax:

```
ZSPC,11,1,<return code>,<filename>,<data area>
```

Return codes:

BLANK operation successful

A	Not two arguments
B	Bad subtype
0	file open failure
N	Nothing to read (nothing in the file or past end of file)

filename

Data location where the name of the file to read has been put.

data area

Data location to read. Exactly one ASCII line will be read, and data area will be cleared beyond the end of the line. A maximum of 950 bytes can be read.

TYPE 11, subtype 2

Writes a line of data to an ASCII file. (MPE/UNIX ONLY)

Syntax:

```
ZSPC,11,2,<return code>,<filename>,<data area>
```

Return code, filename and data area the same as ZSPC 11, subtype 1.

TYPE 11, subtype 3

Reads a line of data from an ASCII file into a TEX question.

Syntax:

```
ZSPC,11,3,<return code>,<filename>,<data area>
```

Return code and filename the same as ZSPC 11, subtype1. Data area must be a TEX question.

TYPE 11, subtype 4

Writes a line of data from a TEX question to an ASCII file.

Syntax:

```
ZSPC,11,4,<return code>,<filename>,<data area>
```

Return code and filename the same as ZSPC 11,1. Data area must be a TEX question.

TYPE 11, subtype 5

Appends a line of data to an existing ASCII file. (MPE ONLY)

Syntax:

```
ZSPC,11,5,<return code>,<filename>,<data area>
```

Return code, filename and data area the same as ZSPC 11, subtype 1. This is useful for dumping detail data to a file from many interviewers.

TYPE 11, subtype 6

Reads a line of data sequentially from an ASCII file.

Syntax:

```
ZSPC,11,6,<return code>,<filename>,<data area>
```

Return code, filename and data area the same as ZSPC 11, subtype 1. The first ZSPC 11,6 will read the first record in the data file, the second SPC,11,6 will read the second record in the data file, etc.

ZSPC TYPE 12

Launches another program from within Survent:

TYPE 12, subtype 1

Runs an external program from within Survent. (MPE ONLY)

Syntax:

```
ZSPC,12,1,<return code>,<programname>,<arguments>
```

Return codes:

BLANK operation successful

A Not two arguments

B CREATEPROCESS failed

C file equation for stdin/stdout failed

programname

Data location of the name of the program to be executed.

arguments

Data locations of arguments to pass to the program. All arguments are passed on at once, and would look the same as it would as an INFO= command.

TYPE 12, subtype 2

Runs an external command from within Survent. (MPE ONLY)

Syntax:

```
ZSPC,12,2,<return code>,<location of command> }
```

Return codes:

BLANK operation successful

A Not two arguments

B CREATEPROCESS failed

C file equation for stdin/stdout failed

location of command

Data location of command to be executed.

Example:

```
{Command: .30 HIDE
COPY fred.asc mary.asc
!VAR }
{Rcode:.1 HIDE
!VAR }
{ZSPC,12,2,Rcode,Command }
```

This would copy the file fred.asc to mary.asc and then continue to the next question.

INDEX

A

C

D

E

F

G

H

I

J

K

L

M

P

Q

R

S

T

V

W

X

Z