SORT-MERGE/XL Programmer's Guide

900 Series HP 3000 Computer Systems

HEWLETT® PACKARD

Manufacturing Part Number: 32650-90884
E0300

U.S.A. March 2000

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Contents

Figures

Tables

Preface

SORT-MERGE/XL Programmer's Guide is intended for use by experienced programmers who are developing applications that require sorting or merging operations. This Programmer's Series manual explains how to use the SORT-MERGE/XL subsystem and related intrinsics.

This manual assumes knowledge of general programming and MPE concepts, but little or no experience with the `HPSORT` or `HPMERGE` intrinsics. For current specific information about the intrinsics, the *MPE/iX Intrinsics Reference Manual*.

Chapter 1, Introduction, offers an overview of the sorting and merging process, with flow charts.

Chapter 2, Creating Core Routines that Sort and Merge, traces the development of Pascal routines that do a simple sort and a simple merge operation. First the example routines are presented step by step with descriptions of the development. The entire examples are presented at the end of the chapter.

Chapter 3, Input and Output by Record, presents examples of a simple sort operation that uses the option of input by record, and a simple merge operation that uses the option of output by record. A Pascal example is also included in this chapter.

Chapter 4, Altering the Collating Sequence, presents an example of using an optional, or alternate, collating sequence. A Pascal example is also included in this chapter.

Chapter 5, Getting SORT-MERGE/XL Information, tells you how to get status information about a completed sorting or merging operation, and how to get title and version information at any point in your sorting or merging program.

Appendix A, HPSORT Intrinsic Status Returns, and Appendix B, HPMERGE Intrinsic Status Returns, list the error number, message, cause, and user corrective action for status returns from the HPSORT and HPMERGE intrinsics.

Appendix C, ASCII/EBCDIC Table, shows the ASCII and EBCDIC character code values, along with their decimal, octal, and hexadecimal equivalents.

Appendix D, FORTRAN Program Examples, gives five FORTRAN programs equivalent to the five Pascal sorting and merging programs used as examples in Chapters 2, 3, and 4.

Appendix E, Data Types, gives tables that show the generic data types used in SORT-MERGE/XL, and their equivalents in HP Business BASIC/XL, HP C/XL, HP COBOL II/XL, HP FORTRAN 77/XL, and HP Pascal/XL.

1 Introduction

This chapter introduces the basic concepts involved in sorting and merging, and provides information about accessing SORT-MERGE/XL programmatically with the sort and merge intrinsics. Topics include:

- Processing a Sort or Merge
- Sorting
- Merging
- Keys
- Error Checking
- Large File Support

For information on using SORT-MERGE/XL as a utility, refer to *SORT-MERGE/XL General User's Guide*.

Processing a Sort or Merge

SORT-MERGE/XL is a set of utilities that allows you to sort a group of records or merge several sorted groups of records into one. The output contains records presented in a specified sequence. You may use SORT-MERGE/XL as a utility and call it from the MPE XL command interpreter (CI), or access it programmatically with the sort and merge intrinsics from a program.

For each sort or merge operation, you must:

Open the work area (Initialization).

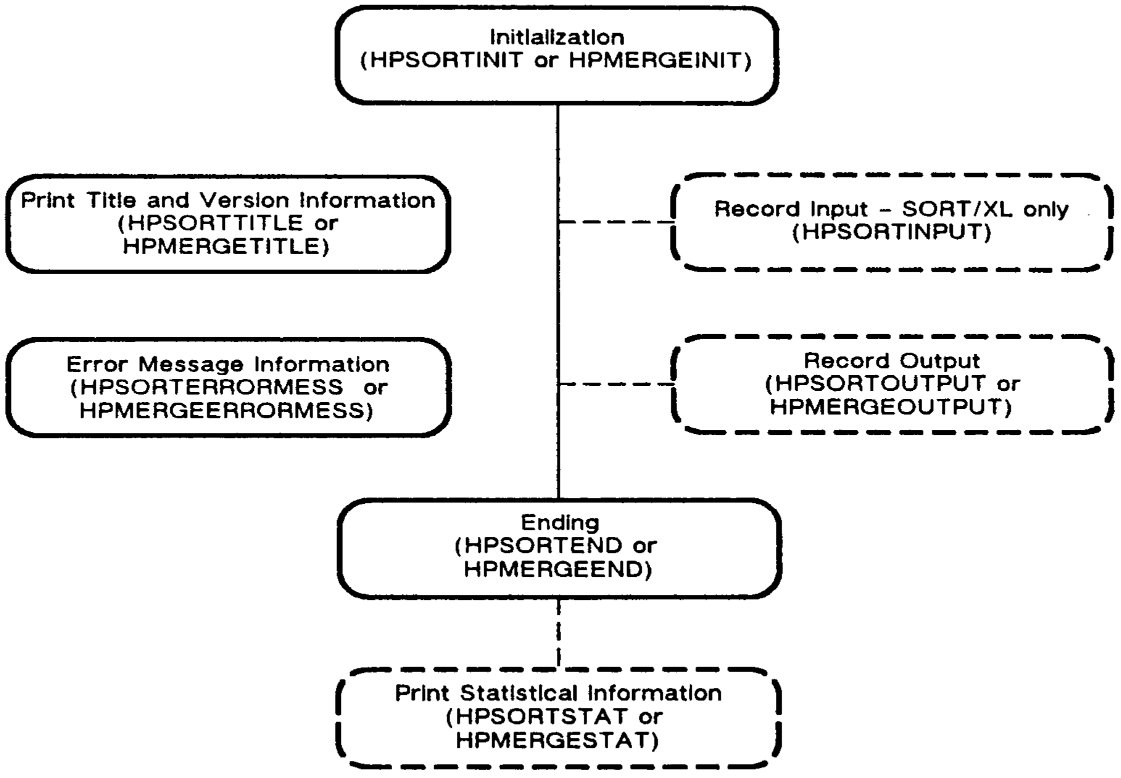
Specify input, output, keys, and collating sequences for the HPSORT-MERGE/XL utility (Initialization).

Close the work area (Ending).

You may check errors and review statistics and title information as well.

Figure 1-1. shows you how HPSORT and HPMERGE intrinsics accomplish these tasks.

Figure 1-1. Flowchart of Intrinsic Order

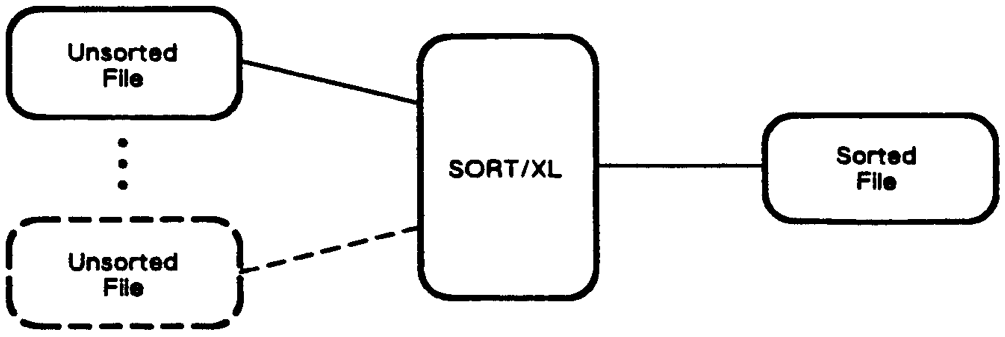


Refer to the *MPE/iX Intrinsics Reference Manual* for detailed information about HPSORT and HPMERGE intrinsics.

Sorting

When you sort a set of records, you must specify the set of keys by which the records are to be ordered. The format of all records must be the same. These records may be submitted to SORT/XL individually or in one or more files. File input (as shown in Figure 1-2.) is the most common method of input. For example, if you input two files, one file containing information on current employees and another file containing information on new employees, you can create a single output file with all the employees sorted alphabetically by last name or numerically by employee number.

Figure 1-2. SORT/XL Operations



The intrinsics used for sorting are:

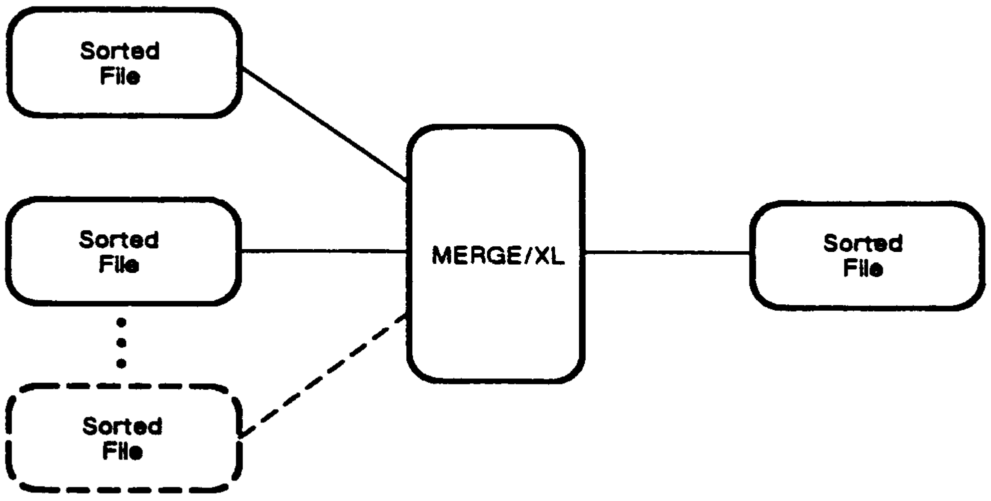
`HPSORTINIT`	Initializes SORT/XL utility and begins the sorting operation (required).
`HPSORTERRORMESS`	Returns error messages.
`HPSORTINPUT`	Allows input to be by record (instead of the usual file input).
`HPSORTOUTPUT`	Allows output to be by record (instead of the usual file input).
`HPSORTEND`	Ends the sorting operation (required).
`HPSORTSTAT`	Prints sorting statistics.
`HPSORTTITLE`	Prints title information for SORT/XL.

NOTE For sorting operations, use only the sort intrinsics. Do not mix sort and merge intrinsics.

Merging

MERGE/XL only merges files that have already been sorted. For example, if you merge a file that contains information on newly hired employees with a file that contains information on current employees, each file must have the same record format and be sorted by the same information. If you merged two such files, each sorted by last name, you would get one output file with all employees sorted by last name. The output file would have the same information and format as the two input files.

Figure 1-3. MERGE/XL Operations



The intrinsics used for merging are:

`HPMERGEINIT`	Initializes MERGE/XL utility and begins the merging operation (required).
`HPMERGEERRORMESS`	Returns error messages.
`HPMERGEOUTPUT`	Allows output to be by record (instead of file).
`HPMERGEEND`	Ends the merging operation (required).
`HPMERGESTAT`	Prints out merging statistics.
`HPMERGETITLE`	Prints out title information for MERGE/XL.

NOTE For merging operations, use only the merge intrinsics. Do not mix sort and merge intrinsics.

Keys

When information is sorted or merged, keys determine the output record sequence. Keys are defined by their beginning position, length, key type, and ordering sequence (ascending or descending).

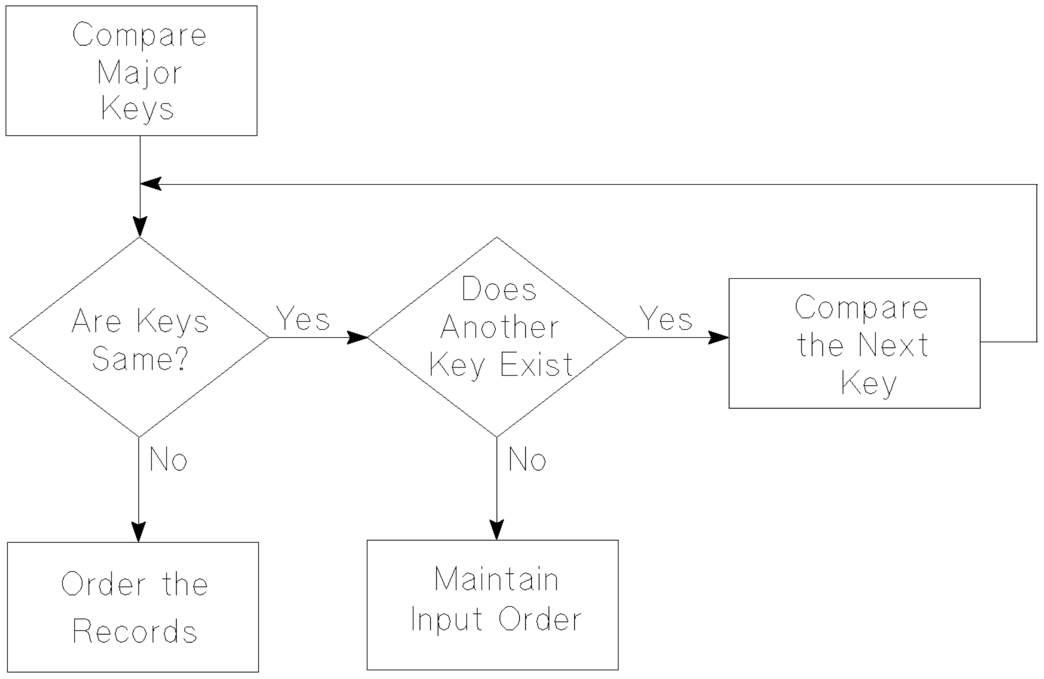
For example, to sort by last names with the record format below, you would specify a key that begins in column 1 and is 20 characters long, byte (ASCII) type, and ascending sequence.

```
Last Name            First Name          Employee Number     Hire Date

Jackson,             Jonathan            000006              06/06/87

0         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890123456789
```

You can specify the collating sequence, which is the order by which the keys are sorted. You can use a predefined order, like ASCII, EBCDIC, or a Native Language, or you can define your own collating sequence.

You may use multiple keys. The major key determines the part of the record examined first in the sort. As Figure 1-4. shows if the major keys of two records are the same, the secondary keys determine the new sequence of the records. If two records have the same first keys and the same second keys, their third keys are compared to determine the sequence. If all the key fields in two or more records are identical, the output file preserves the order of the input records.

Figure 1-4. Key Comparing Operations



For more information about specifying keys, refer to Chapter 2. For more information about collating sequences, refer to Chapter 4.

Error Checking

Good programming practice specifies that each intrinsic call should be checked to ensure that the call was successful. Each intrinsic contains a *status* parameter that returns a value indicating the success or failure of the routine.

If no error occurred with the intrinsic call, a value of zero (0) is returned in the *status* parameter.

If an error did occur with the intrinsic call, the first 16 bits, bits (0:16), of *status* contain the error condition code. The first 16 bits are always negative; all SORT-MERGE/XL errors are fatal. HPSORT and HPMERGE status returns are in Appendixes A and B.

The last 16 bits, bits (16:16), of *status* contain the subsystem identification number. The subsystem number for SORT/XL is 195. The subsystem number for MERGE/XL is 196. (Refer to the *MPE/iX Intrinsics Reference Manual* for information about calculating the error number and subsystem from the *status* parameter.)

The HPMERGEERRORMESS and HPSORTERRORMESS intrinsics return the appropriate error message for each of these error code values. (For information about the use of the error message intrinsics, refer to Chapter 2.)

Large File Support

The SORT/XL subsystem has been modified to support Large File functionality available in release 6.5. SORT/XL can handle multiple input files and the sum of the input file sizes cannot be larger than about 107GB. SORT/XL normally allocates two scratch files each of which is about one hundred twenty percent larger than the sumn of the input files and those files must be smaller than the maximum object size of 128GB.

The maximum native mode data type record length handled by SORT/XL has been raised to about 32,000 bytes from the current limit of 4096 bytes. The actual maximum record length depends on the number and length of the keys.

2 Creating Core Routines That Sort and Merge

This chapter describes two sets of operations: a basic sort and a basic merge. The procedure development is described step by step. The complete examples are at the end of the chapter. These core routines use a minimum set of operations to do a simple sort and merge.

These examples take input from two files and return output to one file. They use the standard ASCII collating sequence. Other options are discussed in other chapters.

Structure of the Core Routines

A simple sorting or merging operation can be performed using only three intrinsics and a subset of their parameters.

- HPSORTINIT begins the sorting process and HPMERGEINIT begins the merging process. Parameters define what will be passed to the utility. All HPSORTINIT and HPMERGEINIT parameters are optional.

  This core routine uses parameters to define the input and output files, output format, record length, keys, and a variable to return error information.

- HPSORTEND and HPMERGEEND close the workspace, ending their processes.
- HPSORTERRORMESS returns the error message associated with the error number returned in the status parameter of HPSORTINIT. HPMERGEERRORMESS returns the error message associated with the error number returned in the status parameter of HPMERGEINIT

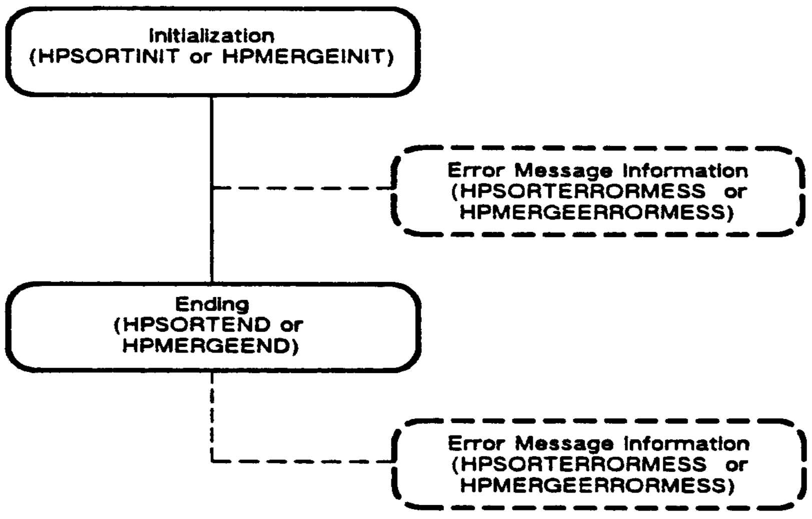
Intrinsic parameters are positional. If a parameter is not specified in an intrinsic call, its position must be maintained by a comma.

NOTE You must initialize and end a sort or a merge within the same procedure. That is, HPSORTEND must be called from the same procedure that called HPSORTINIT, and HPMERGEEND must be called from the same procedure that called HPMERGEINIT.

You may run only one sorting or merging operation at a time. You can not nest them.

Figure 2-1. shows the structure of the core sorting or merging routine.

Figure 2-1. Core Routine



Initializing a Sort or Merge

You must specify the following to initialize the SORT-MERGE/XL utility and start the sorting or merging process:

- input file(s)
- output file
- keys
- data type and collating sequence

The `HPSORTINIT` intrinsic passes the information necessary to initialize the sorting process, and `HPMERGEINIT` passes the information to initialize the merging process.

The syntax for `HPSORTINIT` and `HPMERGEINIT` is:

```
HPSORTINIT (status, inputfiles, outputfile, outputoption,
        reclength, numrecs, numkeys, keys, altseq, keycompare,
        errorproc, statistics, memsize, charseq);

HPMERGEINIT (status, inputfiles, preprocessor, outputfile,
         postprocessor, keysonly, numkeys, keys, altseq,
         keycompare, errorproc, statistics, memsize, charseq);
```

The core routine examples that follow do not use all of the parameters listed above. When you enter these intrinsics, you must maintain the position of any unused parameters with commas.

The *inputfiles* and *outputfiles* parameters in this chapter are files; input and output by record is discussed in Chapter 3. The *altseq* and *charseq* parameters, which alter the collating sequence, are discussed in Chapter 4. The *statistics* parameters, which get SORT-MERGE/XL information, are discussed in Chapter 5

Specifying Input

The most common way to supply information to SORT-MERGE/XL is through an input file. Input considerations include:

- Creating input files.
- Accessing input files.
- Dealing with tape input.

Creating Input Files

SORT-MERGE/XL accepts one or more input files. You create these files with an editor, in a program, or from a database. You may not use `$NULL` as an input file.

Remember that your data items must be in a fixed format. Each data item of the same type must start in the same column. If your data is not in a fixed format, your results will be unpredictable.

Creating Input Files in an Editor You can use any text editor to create fixed format data files stored in character format.

For example, EDIT/3000 keeps your data items lined up by using tabs to separate them. This ensures a fixed format. To follow the example below, enter EDIT/3000, set the tab character to a displayable character, and indicate where you want the tabs to be set.

```
/tabchar = "%", tabs = (21,41,61)
```

To verify that the tab character and the tabs are set the way you want them, enter:

```
/verify tabchar, tabs
```

You will see EDIT/3000 display:

```
tab character = "%"
tabs = (21,41,61)
```

Enter your information in the following manner:

```
    \A
      1       Jones,%Eliza%000001%06/06/87
      2       Smith,%James%000005%06/06/87
      3       Jackson,%Johnathon%000005%06/06/87
      4       Washington,%Lois%000014%07/23/87
      5       Jackson,%Rosa%000022%08/15/87
```

When you list this file, it will appear as follows:

```
Jones,              Eliza               000001              06/06/87
Smith,              James               000002              06/06/87
Jackson,            Johnathon           000003              06/06/87
Washington,         Lois                000004              07/23/87
Jackson,            Rosa                000005              08/15/87
```

Give your file a meaningful name, save it with the KEEP command. This example file contains permanent employee information, so it is kept as PERMEMP.

For more information about EDIT/3000, refer to the *EDIT/3000 Reference Manual*.

Creating Input Files in a Program With intrinsics, you can access SORT-MERGE/XL programmatically from any language. When you create a file from a program, the data can be any data type allowed by the language you are using. For information about creating input files and saving the output file, refer to the programmer's guide for your language, such as:

- *HP Business BASIC/XL Reference Manual*
- *HP C Reference Manual* and *HP C/XL Reference Manual Supplement*
- *COBOL II Reference Manual* and *COBOL II/XL Reference Manual Supplement*
- *HP FORTRAN 77/XL Reference Manual*
- *HP Pascal Reference Manual*

When programming in COBOL, you can sort and merge records directly through the COBOL SORT and MERGE statements. These statements allow you to specify the key, collating sequence, and file or record output to be used by the SORT-MERGE/XL utility. For further information, refer to *COBOL II Reference Manual* and *COBOL II/XL Reference Manual Supplement*.

You will find FORTRAN information in Appendix D of this manual.

Creating Input Files in a Database Creating SORT-MERGE/XL input files from a database depends on your database and access method. For information about creating files and loading information to them from a database, refer to your database manual set.

Accessing the Input File

The SORT and MERGE intrinsics access input files through their file numbers. You pass the identification numbers in the *inputfiles* parameter of the HPSORTINIT or HPMERGEINIT intrinsic. The *inputfiles* parameter is an array. The last number in the array must be zero, as shown in the example below.

To get the file identification numbers, open the input files with the intrinsic HPFOPEN (or FOPEN), as in the example.

The following is from the example at the end of the chapter. The first part opens the files TEMPEMP and PERMEMP, and gets their file numbers, (tempFileNum and permFileNum). The second part puts these identification numbers in an array for the *inputfiles* parameter of the HPSORTINIT or HPMERGEINIT intrinsics.

```
    const
      designator : 2; {HPFOPEN formaldesignator= option}
      domain     : 3; {HPFOPEN file domain= option}
      access     : 11; {used later for output file}
    var
      tempFileNum  : INTEGER;    {HPFOPEN will return}
      permFileNum  : INTEGER;    { with file numbers }
      status       : INTEGER;        {error check}
      tempFile     : packed array [1..10] of CHAR;
      permFile     : packed array [1..10] of CHAR;
      permanent    : INTEGER;

    newFile := '%TEMPEMP%';
    permanent := 1;{file is permanent, in system file domain}
    HPFOPEN (tempFileNum, status, designator, tempFile,
             domain, permanent);

    PermFile := '%PERMEMP%';
    HPFOPEN (permFileNum, status, designator, permFile,
             domain, permanent);
```

Now that you have the file numbers, you can initialize the *inputfiles* parameter:

```
var
  inputfiles   : array [1..3] of INTEGER;
     .
     .
     .
inputfiles[1] :=  tempFileNum;  {from HPFOPEN}
inputfiles[2] :=  permFileNum:  {from HPFOPEN}
inputfiles[3] :=  0;   {last is always zero}
```

If you do not specify anything in *inputfiles* in HPSORTINIT, SORT/XL assumes that input will be by individual record and that you are using the HPSORTINPUT intrinsic. For information about input by record, refer to Chapter 3.

Using Tape Input (SORT/XL Only)

If any of your files are stored on tape, HPSORTINIT needs to know the total number of records that you have on disc and tape. This is specified in the *numrecs* parameter. If you have tape files but do not specify this parameter, SORT/XL defaults to 10,000 records per tape file. SORT/XL takes the size of your disc files from the file label; if all your files are on disc, do not specify this parameter.

Specifying Output

The most common method of maintaining output from SORT-MERGE/XL is by specifying an output file. Output considerations include:

- Creating the Output File.
- Output Record Format (SORT/XL).
- Output Record Format (MERGE/XL).

Creating the Output File

You specify the output file by indicating its file identification number in the *outputfile* parameter in HPSORTINIT or HPMERGEINIT. The *outputfile* parameter is an array. The last number in the array must be zero, as shown in the example below.

To get the file number, open the output file with HPFOPEN (or FOPEN), as in the example.

The following is from the example at the end of the chapter. The first part creates a new permanent file named outFile with WRITE access and with logical records of 80 bytes. The second part puts the number of this file into an array for the *outputfile* parameter of the

HPSORTINIT or HPMERGEINIT intrinsics.

```
    const
      designator = 2; {HPFOPEN formaldesignator= option}
      domain     = 3; {HPFOPEN file domain= option}
      access     = 11; {HPFOPEN access type option #3}
      record_size= 19;  {HPFOPEN record length option #3}
    var
      outFileNum   : INTEGER;
      status       : INTEGER;
      outFile      : packed array [1..10] of CHAR;
      new          : INTEGER;
      write        : INTEGER;
      size         : INTEGER;

    new := 4;  {creates a permanent file}
    write := 1; {file is write-only access}
    size := 80;
    outFile := '%ALLEMP%';
    HPFOPEN (outFileNum, status, designator, outFile,
             domain, new, access, write, record_size, size);
```

Now that you have the file numbers, you can initialize the *inputfiles* parameter:

```
    var
         outputfile   : array [1..2] of INTEGER;
             .
             .
             .
    outputfile[1] :=  outFileNum; {from HPFOPEN}
    outputfile[2] :=  0;
```

If you do not specify anything in *outputfile* SORT-MERGE/XL assumes that output will be by individual record and that you are using the HPSORTOUTPUT intrinsic.

Output Record Format (SORT/XL)

You specify the content of the SORT/XL output file with the *outputoption* parameter of HPSORTINIT. The part of the core routine example at the end of the chapter that specifys the *outputoption* in HPSORTINIT is:

```
    var
      OutputOption : INTEGER;
         .
         .
         .
    OutputOption := 0;  {Output record format same}
                        {  as input record format }
```

The options are *`outputoption`* equals:

0	Output record format is the same as the input record format (default).

1 Each output record contains a 32-bit integer in binary format that indicates the logical record number of the record.

The logical record number assigned to each record is the original record order (the first record is logical record zero, the second record is logical record one, and so on). If you use file input, the files are in the order that they are specified in *`inputfiles`*. For example, if the first file has three records, the logical record number of the fourth record in the second file is 6.

If you use file input, use the FCOPY utility to view the logical record numbers. For example, if your output file is `ALLEMP`, you use the FCOPY utility to print the value to the screen as follows:

```
:FCOPY FROM = ALLEMP; TO= $STDLIST; HEX

HP32212A.03.23 FILE COPIER (C) HEWLETT-PACKARD CO. 1984
```

The first two bytes of the record contain the logical record number. For example, logical record number three (final record number seven, the last of eight output records) is displayed as follows:

```
ALLEMP RECORD 7 (%7, #7)
0000: 0000 0003 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
000C: SAME: TO 0028-1
EOF FOUND IN FROMFILE AFTER RECORD 7
8 RECORDS PROCESSED *** 0 ERRORS

END OF SUBSYSTEM
```

For more information about the FCOPY utility, refer to the *FCOPY Reference Manual* (03000-90064) .

If you use record output, use the `ASCII` intrinsic to convert the logical record number to a readable format. For details about the `ASCII` intrinsic, refer to the *MPE XL Intrinsics Reference Manual*.

2 Each output record contains only the key fields. The primary key is the leftmost; each succeeding key is to the right.

The following example shows the formats of the input and output records if the primary key is last name, the secondary key is employee number. This is the input record format:

```
Jones,           Eliza              000001             06/06/87
```

This is the output format:

```
Jones,                000001
```

3	Each output record contains the logical record number (in binary format) followed by the key fields.

The output format appears the same as for option 2, but the logical record number is stored in the first two bytes of the record. To view the logical record number, use the FCOPY utility or the `ASCII` intrinsic, as in *`outputoption`* = 1.

Output Record Format (MERGE/XL)

You specify the format of the MERGE/XL output file with the *`keysonly`* parameter of `HPMERGEINIT`.

The part of the core routine example at the end of the chapter that specifys the *`keysonly`* in `HPSORTINIT` is:

```
var
  keysonly : INTEGER;
      .
      .
      .
keysonly := 0;  {output record format same as}
                {input record format}
```

The options are *`keysonly`* equals:

0	Output record format is the same as the input record format (default).
1	Each output record contains only the key fields. The primary key is the leftmost; each succeeding key is to the right.

The following example shows the record format for input and output records if the primary key is last name and the secondary key is employee number (*`keysonly`* =1). This is the input format:

```
Jones,          Eliza              000001              06/06/87
```

This is the output format:

```
Jones,                000001
```

Specifying Keys

The key indicates the part of each record that is compared to determine output record order. In `HPSORTINIT` or `HPMERGEINIT`, you indicate the number of keys with the *`numkeys`* parameter and give key information in the *keys* parameter.

The part of the core routine example at the end of the chapter that specifies keys is:

```
    var
      num_keys :  INTEGER;
      keys : array [1..4] of INTEGER;
         .
         .
         .
    num_keys := 1;
    keys [1] := 1   {key begins}
    keys [2] := 20; {key length}
    keys [3] := 0;  {byte data }
    keys [4] := 0;  {ascending order}
```

The *keys* parameter is an integer array; each key contains four elements you specify. The elements are:

- Beginning byte position of the key.
- Number of bytes in the key.
- Ordering sequence (0 for ascending, 1 for descending)
- Type of data to be sorted by the key. See types below:

NOTE Language equivalents of the following data types are given in Appendix E.

0 Byte (usually used).

If the key is ASCII or EBCDIC, it is byte type. With byte type, element 2 of *keys* contains the number of characters in the key.

1 twos complement format (integer).

This type is used for shortint (2-byte) or integer (4-byte) keys.

2 HP 3000 floating point.

This type is used for data that is in MPE V/E floating point format. This type is used for 4-byte and 8-byte real numbers.

3 IEEE standard floating point.

This type is used for data in the MPE XL floating point format. This type is used for 4-byte, 8-byte, or 16-byte real numbers.

4 Packed decimal with odd number of digits.

This is a COBOL data type.

5 Packed decimal with even number of digits.

This is a COBOL data type.

6 Display trailing sign.

	This is a COBOL data type.
7	Display leading sign.
	This is a COBOL data type.
8	Display trailing sign separate.
	This is a COBOL data type.
9	Display leading sign separate.
	This is a COBOL data type.
10	Character.
	This data type is used in Native Language situations, when you have character data (as in option 0), but will be using a Native Language collating sequence.
11	*Reserved*.
12	Short floating point decimal.
	This is an HP Business BASIC data type.
13	Floating point decimal.
	This is an HP Business BASIC data type.

NOTE The integrity of the *keys* array must be maintained throughout the sorting or merging operation. Do not change *keys* until after you have called HPSORTEND or HPMERGEEND.

Specifying Data and Sequence

If your key is a character or byte data type, you may use the *altseq* parameter to specify two things.

- The data type of your input key.
- The collating sequence (sort order) you want the sort to follow.

The *altseq* parameter is an array with two parts. The first element of the array is defined by the following table.

		Collating Sequence:		
		ASCII	**EBCDIC**	**ALTERNATE**
Data Type:	**ASCII**	CHR(255)	CHR(2)	CHR(0)
	EBCDIC	CHR(1)	CHR(255)	Undefined

The second element of *altseq* specifies the number of characters in the collating sequence. Remember that SORT-MERGE/XL begins counting on zero, and so you subtract one from the ordinary counting number. For example, ASCII has 256 characters, so the example below specifies the value 255.

The example at the end of this chapter uses ASCII input and ASCII sequence. The following is the part of the example that specifies the input data type and sorting sequence.

```
var
  altseq : array [1..2] of CHAR;
     .
     .
     .
altseq[1] := CHR(255); {data = ASCII,   }
                       {sequence = ASCII}
altseq[2] := CHR(255); {256 characters in ASCII}
```

The *altseq* parameter is used only for byte data types The most common are ASCII and EBCDIC. Refer to Chapter 4 for information about alternate collating sequences, like one you design yourself or one of the predefined Native Language types.

Using HPSORTINIT

The following code segment demonstrates the the HPSORTINIT intrinsic. The parameters used are those already discussed.

```
var
  status       : INTEGER;
  inputfiles   : array [1..3] of INTEGER;
  outputfile   : array [1..2] of INTEGER;
  outputoption : INTEGER;
  NumKeys      : INTEGER;
  keys         : array [1..4] of INTEGER;
  altseq       : array [1..2] of CHAR;

inputfiles[1] :=  tempFileNum; {from HPFOPEN}
inputfiles[2] :=  permFileNum: {from HPFOPEN}
inputfiles[3] :=  0;

outputfile[1] :=  outFileNum; {from HPFOPEN}
outputfile[2] :=  0;

outputOption := 0;  {output record format same}
                    { as input record format  }

numKeys := 1;  {one key}
keys[1] := 1;  {key begins}
keys[2] := 20; {key length}
keys[3] := 0;  {byte data}
keys[4] := 0;  {ascending order}

altseq[1] := CHR(255); {data = ASCII and}
                       {sequence = ASCII}
altseq[2] := CHR(255); {256 characters in ASCII}

HPSORTINIT (status, inputfiles, outputfile,
            outputOption,,, numKeys, keys, altseq,,,,,);
```

Using HPMERGEINIT

The following code segment demonstrates the HPMERGEINIT intrinsic. The parameters used are those already discussed.

Note that the HPMERGEINIT intrinsic call is different from the HPSORTINIT intrinsic call.

```
var
  status       : INTEGER;
  inputfiles   : array [1..3] of INTEGER;
  outputfile   : array [1..2] of INTEGER;
  keysonly     : INTEGER;
  numKeys      : INTEGER;
  keys         : array [1..4] of INTEGER;
  altseq       : array [1..2] of CHAR;

inputfiles[1] :=  tempFileNum; {from HPFOPEN}
inputfiles[2] :=  permFileNum: {from HPFOPEN}
inputfiles[3] :=  0;

outputfile[1] :=  outFileNum;  {from HPFOPEN}
outputfile[2] :=  0;

keysonly  := 0;  {output record format same as }
                 {input record format          }

numKeys := 1;  {one key}
keys[1] := 41;  {key begins}
keys[2] := 20; {key length}
keys[3] := 0;  {byte data}
keys[4] := 0;  {ascending order}

altseq[1] := CHR(255); {data = ASCII           }
                       {sequence = ASCII       }
altseq[2] := CHR(255); {256 characters in ASCII}

HPMERGEINIT (status, inputfiles, outputfile,
             keysonly,,, numkeys, keys, altseq,,,,,);
```

Ending Sorting or Merging

You end the sorting or merging process with HPSORTEND or HPMERGEEND.

The syntax for HPSORTEND and HPMERGEEND is:

```
HPSORTEND (status, statistics);

HPMERGEEND (status, statistics);
```

The part of the core routine example at the end of the chapter that uses the HPSORTEND intrinsic follows.

```
var
  status : INTEGER
      .
      .
      .
HPSORTEND (Status, );
```

Error Checking

The *status* parameter of HPSORTINIT and HPMERGEINIT is a 32-bit integer variable that returns the status of each sort or merge intrinsic call. If no errors or warnings are encountered, *status* returns 32 bits of zero. If errors are encountered, *status* is interpreted as two 16-bit fields. Bits (0:16), the leftmost halfword, is *status.info*. A negative value here indicates an error condition. Bits (16:16) are *status.subsys*. A number here is coded to indicate the subsystem where the error occurred. A value of 195 indicates SORT/XL. A value of 196 identifies MERGE/XL. Although *status* is an optional parameter, it is good programming practice to specify it. If it is not specified and an error occurs, the calling process will abort.

You use HPSORTERRORMESS and HPMERGEERRORMESS to display a descriptive message to the user if errors occur during a sort or merge. The HPSORTERRORMESS and HPMERGEERRORMESS *status* parameter accepts the error number returned from the HPSORTINIT and HPMERGEINIT *status* parameter. Other error message parameters return the text message associated with that error number, and the length of the message.

The syntax of HPSORTERRORMESS and HPMERGEERRORMESS is:

```
HPSORTERRORMESS (status, message, length);

HPMERGEERRORMESS (status, message, length);
```

The HPSORTERRORMESS and HPMERGEERRORMESS expect to interpret errors, not successes. The merge and sort *status* parameters return zero if no errors occurred. If you do not check this value before calling HPSORTERRORMESS or HPMERGEERRORMESS as the example below does, the result could be confusing. Passing zero to the error message intrinsics *status* parameter, causes the *message* parameter to return the message: "HPERRORMSG failed inside HPSORTERRORMESS" or "HPERRORMSG failed inside HPMERGEERRORMESS".

If the HPSORTERRORMESS or HPMERGEERRORMESS intrinsic call fails, *status* returns the error number associated with that failure.

The error numbers and messages for sort and merge intrinsics are listed in Appendixes A and B.

For more information about calculating *status.info* and *status.subsys*, refer to the *MPE XL Intrinsics Reference Manual*.

The following is part of the core sort example, and shows how HPSORTERRORMESS is used.

```
var
  status  : INTEGER;    {from a HPSORT intrinsic call}
  message : array [1..80] of CHAR;
  length  : INTEGER;

If status <> 0 then
   begin
      message := '';
      HPSORTERRORMESS (status, message, length);
      writeln (message)
   end;
```

Example of Core Sorting Routine

The following program sorts the personnel files shown below. They are sorted together by last name. The record size is determined by the input files. The *status* parameter is checked after the calls to HPSORTINIT and HPSORTEND.

The files that are used in this example are as follows (character positions and data descriptions are indicated for convenience only):

TEMPEMP   Information file about temporary employees:

```
Last Name           First Name       Employee Number      Hire Date

Gangley,            Tomas               000003             06/06/87
Rields,             Evelyn              000007             07/12/87
Everett,            Joyce               000029             10/19/87

0        1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
```

PERMEMP   Information file about permanent employees:

```
Last Name          First Name        Employee Number      Hire Date

Jones,              Eliza               000001             06/06/87
Smith,              James               000005             06/06/87
Jackson,            Johnathon           000006             06/06/87
Washington,         Lois                000014             07/23/87
Jackson,            Rosa                000022             08/15/87

0        1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
```

Example 2-1. SORTFILE Program

```
   program SORTFILE (input,output);

   {This program reads the files, TEMPEMP and PERMEMP, }
   {sorts by last name, and outputs to the file, ALLEMP}

   var
      tempFileNum: INTEGER;
      permFileNum: INTEGER;
      outFileNum : INTEGER;
      status     : INTEGER;

   procedure HPFOPEN  ; intrinsic;
   procedure HPSORTINIT; intrinsic;
   procedure HPSORTERRORMESS; intrinsic;
   procedure HPSORTEND; intrinsic;
   procedure FCLOSE; intrinsic;


   procedure OPEN_FILES;

   const
      designator  = 2;
      domain      = 3;
      access      = 11;
      record_size = 19;

   var
      tempFile  : packed array [1..10] of CHAR;
      permFile  : packed array [1..10] of CHAR;
      outFile   : packed array [1..10] of CHAR;
      permanent : INTEGER;
      new       : INTEGER;
      write     : INTEGER;
      size      : INTEGER;

   begin
      tempFile := '%TEMPEMP%';
      permanent := 1;
      HPFOPEN (tempFileNum, status, designator, tempFile, domain, permanent)
;

      permFile := '%PERMEMP%';
      HPFOPEN (permFileNum, status, designator, permFile, domain, permanent)
;     new := 4;
      write := 1;
      size := 80;
      outFile := '%ALLEMP%';
      HPFOPEN (outFileNum, status, designator, outFile, domain, new, access,
               write, record_size, size);
   end;
```

```
procedure DO_SORT;

var
   inputfiles   : array [1..3] of INTEGER;
   outputfile   : array [1..2] of INTEGER;
   outputOption : INTEGER;
   numKeys      : INTEGER;
   keys         : array [1..4] of INTEGER;
   altseq       : packed array [1..2] of CHAR;
   message      : packed array [1..80] of CHAR;
   length       : INTEGER;
   statistics   : array [1..6] of INTEGER;

begin
   inputfiles [1] := tempFileNum;
   inputfiles [2] := permFileNum;
   inputfiles [3] := 0;
   outputfile [1] := outFileNum;                              {from
HPFOPEN}
   outputfile [2] := 0;
   outputOption := 0;   {output record format same as input record format}
   numKeys := 1;                                                {one key   }
   keys[1] := 1;                                                {key begins}
   keys[2] := 20;                                               {key length}
   keys[3] := 0;                                                {byte data }
   keys[4] := 0;                                           {ascending order}
   altseq[1] := CHR(255);                     {data = ASCII; sequence = ASCII}
   altseq[2] := CHR(255);                           {256 characters in ASCII}
   HPSORTINIT (status, inputfiles, outputfile, outputOption,,, numKeys, keys,
               altseq,,,,,);
   if status <> 0 then                    {If error in HPSORTINIT            }
   begin                                 {Get message and write it to screen}
      Message := ' ';
      HPSORTERRORMESS (status, message, length);
      writeln (message);
   end;

   HPSORTEND (status, );

   if status <> 0 then                    {If error in HPSORTEND             }
   begin                                 {Get message and write it to screen}
      message := ' ';
      HPSORTERRORMESS (status, message, length);
      writeln (message)
   end;
end;


procedure CLOSE_FILES;

var
   disposition : SHORTINT;
```

```
      securityCode : SHORTINT;

   begin
      disposition := 0;
      securityCode := 0;
      FCLOSE (tempFileNum, disposition, securityCode);
      FCLOSE (permFileNum, disposition, securityCode);
      disposition := 1;
      FCLOSE (outFileNum, disposition, securityCode);
   end;

   begin {main}
      OPEN_FILES;
      DO_SORT;
      CLOSE_FILES;
   end.
```

When this program is executed, the output from the sort is written to ALLEMP. To view the output:

```
:print allemp

Everett,            Joyce               000029              10/19/87
Gangley,            Tomas               000003              06/06/87
Jackson,            Jonathan            000006              06/06/87
Jackson,            Rosa                000022              08/15/87
Jones,              Eliza               000001              06/06/87
Rields,             Evelyn              000007              07/12/87
Smith,              James               000005              06/06/87
Washington,         Lois                000014              07/23/87
```

Example of Core Merging Routine

The following program merges the personnel files shown at the beginning of the previous example. The input files are already sorted by employee number.

In this example, the files are merged by employee number. The record size is determined by the input files. The *status* parameter is checked after the calls to HPMERGEINIT and HPMERGEEND.

Example 2-2. MERGEFILE Program

```
    program MERGEFILE (input,output);

   {This program reads the files, TEMPEMP and PERMEMP, merges}
   {them by employee number, and outputs to the file, ALLEMP }

   var
      tempFileNum: INTEGER;
      permFileNum: INTEGER;
      outFileNum : INTEGER;
      status     : INTEGER;

   procedure HPFOPEN  ; intrinsic;
   procedure HPMERGEINIT; intrinsic;
   procedure HPMERGEERRORMESS; intrinsic;
   procedure HPMERGEEND; intrinsic;
   procedure FCLOSE; intrinsic;


   procedure OPEN_FILES;

   const
      designator  = 2;
      domain      = 3;
      access      = 11;
      record_size = 19;

   var
      tempFile  : packed array [1..10] of CHAR;
      permFile  : packed array [1..10] of CHAR;
      outFile   : packed array [1..10] of CHAR;
      permanent : INTEGER;
      new       : INTEGER;
      write     : INTEGER;
      size      : INTEGER;
       tempFile := '%TEMPEMP%';
       permanent := 1;
      HPFOPEN (tempFileNum, status, designator, tempFile, domain, permanent)
;

       permFile := '%PERMEMP%';
      HPFOPEN (permFileNum, status, designator, permFile, domain, permanent)
;
       new := 4;
       write := 1;
   size := 80;
   outFile := '%ALLEMP%';
   HPFOPEN (oOutFileNum, status, designator, outFile, domain, new, access,
            write, record_size, size);
end;


procedure DO_MERGE;

var
   inputfiles   : array [1..3] of INTEGER;
   outputfile   : array [1..2] of INTEGER;
   keysonly     : INTEGER;
   numKeys      : INTEGER;
   keys         : array [1..4] of INTEGER;
   altseq       : packed array [1..2] of CHAR;
   message      : packed array [1..80] of CHAR;
   length       : INTEGER;

begin
   inputfiles [1] := tempFileNum;
   inputfiles [2] := permFileNum;
   inputfiles [3] := 0;

   outputfile [1] := outFileNum;                              {from HPFOPEN}
   outputfile [2] := 0;

   keysonly := 0;      {output record format same as input record format }
  numKeys := 1;                                                   {one key}
  keys[1] := 41;                                               {key begins}
  keys[2] := 20;                                               {key length}
  keys[3] := 0;                                                 {byte data}
  keys[4] := 0;                                           {ascending order}

   altseq[1] := CHR(255);                {data = ASCII; sequence = ASCII}
   altseq[2] := CHR(255);                        {256 characters in ASCII}

   HPMERGEINIT (status, inputfiles,, outputfile,, keysonly, numKeys, keys,
                altseq,,,,,);
   if status <> 0 then                {if error in HPMERGEINIT           }
  begin                               {get message and print it to screen}
     message := ' ';
     HPMERGEERRORMESS (status, message, length);
     writeln (Message);
   end;

   HPMERGEEND (status, statistics);

  if status <> 0 then                  {if error in HPMERGEEND            }
  begin                               {get message and print it to screen}
     message := ' ';
     HPMERGEERRORMESS (status, message, length);
     writeln (message)
   end;
end;
```

```
procedure CLOSE_FILES;

var
   disposition : SHORTINT;
   securityCode : SHORTINT;

begin
   disposition := 0;
   securityCode := 0;
   FCLOSE (tempFileNum, disposition, securityCode);
   FCLOSE (permFileNum, disposition, securityCode);
   disposition := 1;
   FCLOSE (outFileNum, disposition, securityCode);

end;


begin {main}
   OPEN_FILES;
   DO_MERGE;
   CLOSE_FILES;
end.
```

When this program is executed, the output is written to ALLEMP. To view ALLEMP:

```
:print allemp

Jones,              Eliza               000001              06/06/87
Gangley,            Tomas               000003              06/06/87
Smith,              James               000005              06/06/87
Jackson,            Jonathan            000006              06/06/87
Rields,             Evelyn              000007              07/12/87
Washington,         Lois                000014              07/23/87
Jackson,            Rosa                000022              08/15/87
Everett,            Joyce               000029              10/19/87
```

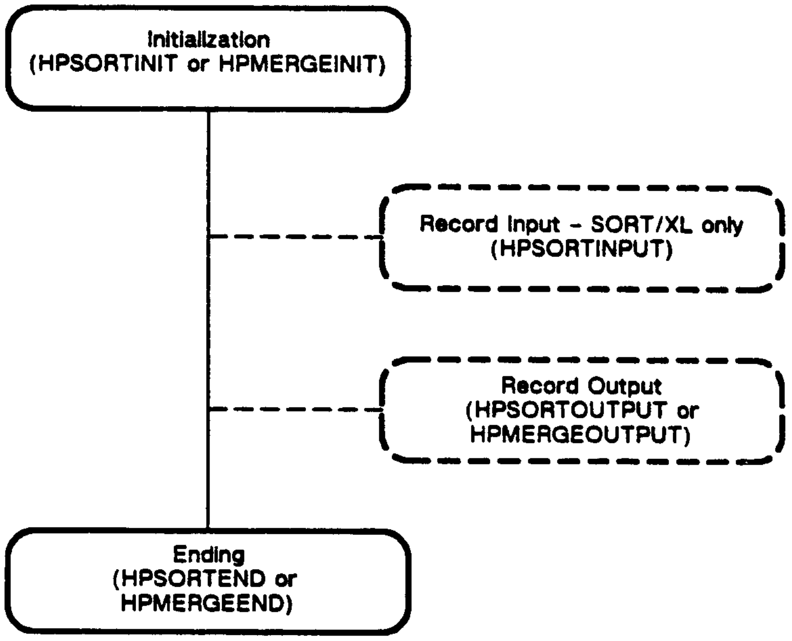
3 Input and Output by Record

Record input and output are optional I/O methods for SORT/XL and MERGE/XL. This chapter presents a general overview of the record options. Next, the intrinsics used for input by record are described, and an example is provided. Last, the intrinsics for output by record are described, and an example provided. (The default I/O methods, input and output by files, are discussed in Chapter 2.)

Using Record Input and Output

As shown in Figure 3-1., record input and output are processed after SORT-MERGE/XL is intialized and before SORT-MERGE/XL is ended.

Figure 3-1. Record Input and Output



You can use record input or record output when you wish to alter the records.

Choose the input by record option when you want to sort a set of input that is available to you one record at a time. This happens when the records you want to sort are created within the sorting program or when records are intended to be received interactively (for example, if your program accepts input from a terminal). Record input is only available for SORT/XL.

Choose the output by record option when you want to sort or merge information during a program's execution without directly storing the information. For example, use it if you want to display a sorted list to a user's terminal, or if you want a chance to modify the output before storing in a file.

Record Input

Record input is useful for sorting information that is entered interactively or produced internally by a program. To input records to SORT/XL you use the `HPSORTINPUT` intrinsic.

MERGE/XL does not allow record input. The syntax of `HPSORTINPUT` is:

```
HPSORTINPUT (status, buffer, length);
```

The *buffer* parameter contains the record to be input.

NOTE The *buffer* parameter is a character (byte) array. If your record contains other data types, you must build *buffer* byte by byte.

The *length* parameter contains the length of the record.

NOTE This value must not be greater than the value of the *reclength* parameter in the `HPSORTINIT` intrinsic.

An example of using `HPSORTINPUT` follows:

```
var
  status : INTEGER;
  buffer : packed array [1..80] of CHAR;
  length : INTEGER;
     .
     .
     .
HPSORTINPUT (status, buffer, length);
```

To use record input:

- The *reclength* parameter in `HPSORTINIT` *must* be specified.
- The *inputfiles* parameter in `HPSORTINIT` *must not* be specified.

Example of Record Input

The following program sorts the personnel files shown below. They are sorted by last name. The program marks the employee numbers of the temporary employees with an asterisk.

These two files, TERMEMP and PERMEMP, are used in the following example. (The data descriptions in the top line, and the character positions along the bottom do not appear in the file. They are for your convenience only.)

TEMPEMP Information file about temporary employees:

```
   Last Name           First Name       Employee Number      Hire Date

   Gangley,            Tomas               000003              06/06/87
   Rields,             Evelyn              000007              07/12/87
   Everett,            Joyce               000029              10/19/87

   0        1         2         3         4         5         6         7
   1234567890123456789012345678901234567890123456789012345678901234567890
```

PERMEMP Information file about permanent employees:

```
   Last Name          First Name        Employee Number      Hire Date

   Jones,              Eliza               000001              06/06/87
   Smith,              James               000005              06/06/87
   Jackson,            Johnathon           000006              06/06/87
   Washington,         Lois                000014              07/23/87
   Jackson,            Rosa                000022              08/15/87

   0        1         2         3         4         5         6         7
   1234567890123456789012345678901234567890123456789012345678901234567890
```

Example 3-1. SORTREC_IN Program

```
program SORTREC_IN (input,output);
{This program reads the files, TEMPEMP & PERMEMP, alters the TEMPEMP records, }
{passes all records to SORT/XL, and outputs to the file, ALLEMP.              }

var
   tempFileNum: INTEGER;
   permFileNum: INTEGER;
   outFileNum : INTEGER;
   status     : INTEGER;

procedure HPFOPEN; intrinsic;
procedure HPSORTINIT; intrinsic;
function  FREAD : SHORTINT; intrinsic;
procedure HPSORTINPUT; intrinsic;
procedure HPSORTEND; intrinsic;
procedure FCLOSE; intrinsic;


procedure OPEN_FILES;
```

```
const
   designator  = 2;
   domain      = 3;
   access      = 11;
   record_size = 19;

var
   tempfile  : packed array [1..10] of CHAR;
   permfile  : packed array [1..10] of CHAR;
   outfile   : packed array [1..10] of CHAR;
   permanent : INTEGER;
   new       : INTEGER;
   write     : INTEGER;
   size      : INTEGER;

begin
   tempfile := '%TEMPEMP%';
   permanent := 1;
   HPFOPEN (tempFileNum, status, designator, tempfile, domain, permanent) ;

   permfile := '%PERMEMP%';
   HPFOPEN (permFileNum, status, designator, permfile, domain, permanent) ;
   new := 4;
   write := 1;
   size := 80;
   outfile := '%ALLEMP%';
   HPFOPEN (outFileNum, status, designator, outfile, domain, new, access,
            write, record_size, size);

procedure DO_SORT;

var
   outputfile   : array [1..2] of INTEGER;
   outputOption : INTEGER;
   numKeys      : INTEGER;
   keys         : array [1..4] of INTEGER;
   altseq       : packed array [1..2] of CHAR;
   message      : packed array [1..80] of CHAR;
   length       : INTEGER;
   lngth        : SHORTINT;
   buffer       : packed array [1..80] of CHAR;
   recLength    : INTEGER;

begin
   outputfile [1] := outFileNum;                                {From HPFOPEN}
   outputfile [2] := 0;
   outputOption := 0;       {output record format same as input record format }
   recLength := 80;                                     {maximum record length}
   numKeys := 1;                                                      {one key}
   keys[1] := 1;                                                   {key begins}
   keys[2] := 20;                                                  {key length}
   keys[3] := 0;                                                    {byte data}
   keys[4] := 0;                                              {ascending order}
   altseq[1] := CHR(255);                       {data = ASCII; sequence = ASCII}
   altseq[2] := CHR(255);                              {256 characters in ASCII}
end;

   HPSORTINIT (status,, outputfile, outputOption,
      recLength,, numKeys, keys, altseq,,,,,);

    length := 72;                                         {read 72 characters}

    repeat                           {read temporary employee file by record}
       Lngth := FREAD (TempFileNum, Buffer, Length);
       Buffer[40] := '*';                                    {Mark the record}
       if Lngth <> 0 then
          HPSORTINPUT (Status, Buffer, Length);
    until Lngth = 0;

    repeat                            {read permanent employee file by record}
      lngth := FREAD (permfilenum, buffer, length);
      if lngth <> 0 then
         HPSORTINPUT (status, buffer, length)
    until lngth = 0;

   HPSORTEND (status, );
end;

procedure CLOSE_FILES;

var
   disposition : SHORTINT;
   securityCode : SHORTINT;

begin
   disposition := 0;
   securityCode := 0;
   FCLOSE (tempFileNum, disposition, securityCode);
   FCLOSE (permFileNum, disposition, securityCode);
   disposition := 1;
   FCLOSE (outFileNum, disposition, securityCode);
end;

begin {main}
   OPEN_FILES;
   DO_SORT;
   CLOSE_FILES;
end.
```

When this program is executed, the output is written to ALLEMP. To view ALLEMP:

```
   :print allemp

   Everett,            Joyce              *000029             10/19/87
   Gangley,            Tomas              *000003             06/06/87
   Jackson,            Jonathan            000006             06/06/87
   Jackson,            Rosa                000022             08/15/87
   Jones,              Eliza               000001             06/06/87
   Rields,             Evelyn             *000007             07/12/87
   Smith,              James               000005             06/06/87
   Washington,         Lois                000014             07/23/87
```

Record Output

Record output is useful when you want to create temporary output of a set of sorted or merged data, to create a subset of the sorted or merged information, or to alter the records before outputting them.

To output from SORT-MERGE/XL by record, use either the `HPSORTOUTPUT` or `HPMERGEOUTPUT`intrinsic. The syntax is:

```
HPSORTOUTPUT (status, buffer, length);

HPMERGEOUTPUT (status, buffer, length);
```

The *buffer* parameter contains the record to be input.

NOTE The *buffer* parameter is a character (byte) array. If your record contains other data types, you must reconstruct them from *buffer*.

The *length* parameter returns the length of the record. After the last record is reached, *length* returns a value of negative one (-1).

If you use record output, *do not* specify the *outputfile* parameter in `HPSORTINIT` or `HPMERGEINIT`. Use the intrinsic `ASCII` if you have specified *outputoption* in `HPSORTINIT` to be binary format, but you want displayable ASCII characters instead.

An example follows:

```
var
  status : INTEGER;
  buffer : packed array [1..80] of CHAR;
  length : SHORTINT;
      .
      .
      .
HPSORTOUTPUT (status, buffer, length);
```

Example of Record Output

The following program sorts the personnel files, TEMPEMP and PERMEMP, that were also used for the last example. They are sorted by last name. The output records are altered before they are sent to $STDLIST.

Example 3-2. SORTREC_OUT Program

```
program SORTREC_OUT (input,output);

{This program reads data from the TEMPEMP and PERMEMP files, sorts them by last
{name, outputs them by record, alters the output record, and prints the record to
($STDLIST.}

var
   tempFileNum: INTEGER;
   permFileNum: INTEGER;
   status     : INTEGER;

procedure HPFOPEN  ; intrinsic;
procedure HPSORTINIT; intrinsic;
procedure HPSORTOUTPUT; intrinsic;
procedure HPSORTEND; intrinsic;
procedure FCLOSE; intrinsic;
procedure OPEN_FILES;

const
   designator = 2;
   domain     = 3;
   access     = 11;

var
   tempfile  : packed array [1..10] of CHAR;
   permfile  : packed array [1..10] of CHAR;
   permanent : INTEGER;

begin
   tempfile := '%TEMPEMP%';
   permanent := 1;
   HPFOPEN (tempFileNum, status, designator, tempfile, domain, permanent);

   permfile := '%PERMEMP%';
   HPFOPEN (permFileNum, status, designator, permfile, domain, permanent);
end;

procedure DO_SORT;

var
   inputfiles   : array [1..3] of INTEGER;
   outputOption : INTEGER;
   numKeys      : INTEGER;
   keys         : array [1..4] of INTEGER;
   altseq       : packed array [1..2] of CHAR;
   message      : packed array [1..80] of CHAR;
   length       : INTEGER;
   buffer       : packed array [1..80] of CHAR;

begin
   inputfiles [1] := tempFileNum;
   inputfiles [2] := permFileNum;
   inputfiles [3] := 0;

   outputOption := 0;        {output record format same as input record format}

   numKeys := 1;                                                       {one key}
   keys[1] := 1;                                                    {key begins}
   keys[2] := 20;                                                   {key Length}
   keys[3] := 0;                                                     {byte data}
   keys[4] := 0;                                               {ascending order}

   altseq[1] := CHR(255);                      {data = ASCII; sequence = ASCII}
   altseq[2] := CHR(255);                             {256 characters in ASCII}

   HPSORTINIT (status, inputfiles,, outputOption,,,
      numKeys, keys, altseq,,,,,);

   repeat                                      {get output record and alter it}
      HPSORTOUTPUT (status, buffer, length);
      strmove (7, 'Empl. #',1, buffer, 33);
      strmove (10, 'Hire Date:', 1, buffer, 50);
      if length >0 then
         writeln (buffer);
   until length <0;

   HPSORTEND (status, );
end;

procedure CLOSE_FILES;

var
   disposition : SHORTINT;
   securityCode : SHORTINT;

begin
   disposition := 0;
   securityCode := 0;
   FCLOSE (tempFileNum, disposition, securityCode);
   FCLOSE (permFileNum, disposition, securityCode);
end;

begin {main}
   OPEN_FILES;
   DO_SORT;
   CLOSE_FILES;
end.
```

When this program is executed, the output is written to the screen:

```
   Everett,            Joyce        Empl. # 000029   Hire Date: 10/19/87
   Gangley,            Tomas        Empl. # 000003   Hire Date: 06/06/87
   Jackson,            Jonathan     Empl. # 000006   Hire Date: 06/06/87
   Jackson,            Rosa         Empl. # 000022   Hire Date: 08/15/87
```

```
Jones,              Eliza       Empl. # 000001   Hire Date: 06/06/87
Rields,             Evelyn      Empl. # 000007   Hire Date: 07/12/87
Smith,              James       Empl. # 000005   Hire Date: 06/06/87
Washington,         Lois        Empl. # 000014   Hire Date: 07/23/87
```

4 Altering the Collating Sequence

This chapter describes choosing an optional, or alternate, collating sequence. The chapter begins with a discussion of when or why you would want to do this. The middle of the chapter tells you how to specify alternate ASCII sequences, and concludes with a Pascal example. The chapter ends with directions for NL (native language) collating sequences.

Sorting and Arranging

When you begin a sort, you must specify the keys, the place where records will be compared. Then, the keys of each record are compared and put into a hierarchichal arrangment. The resulting arrangement depends two things: the sequence and the order. You must specify both.

The sequence determines what follows what: for example, 0,1,2,...,9 in digits. There are predefined sequences, like the standard alphanumeric ones, or you can create your own.

Sequencing depends on the data type of the key. You pass the type in one field of the *keys* parameter. Number type keys are sorted in standard numeric order. ASCII or EBCDIC byte types (type 0 in the *keys* parameter) are sorted in their own sequences, shown in Appendix C.

You can, however, vary the sequence of byte types if you wish. You may want an alphabetic sequence that is not case sensitive. For example, you may want "AaBbCc..." instead of the ASCII sequence of "ABC ... abc". You may want your EBCDIC data sorted ignoring hyphens and punctuation, or with the digits before the letters.

The order determines which end of the sequence will come first. There are two options: ascending or descending. For example, the digits in ascending order are: 0,1,2,...,9; the digits in descending order are: 9,8,7,...,0.

NOTE When performing Native Language (NL) functions, alphanumeric sorting and merging needs to be done using the specifics of the native language. See the end of this chapter for more information.

Altering the Sequence

You can specify an alternate sequence in the *`altseq`* parameter of the `HPSORTINIT` or `HPMERGEINIT` intrinsic. (The default sequence is ASCII.) Only ASCII type is allowed to request alternate sequences; when you specify an alternate collating sequence, you must set the *`keys`* parameter data type to zero (byte data).

The *`altseq`* parameter accepts a array.

The first element of the array is set by the following table. The rows of the table show the type of data you are sorting, and the columns of the table show the type of sequence you want. For alternate sequences, set the first element of the *`altseq`* parameter to CHR(0).

		Collating Sequence:		
		ASCII	**EBCDIC**	**ALTERNATE**
Data Type:	**ASCII**	`CHR(255)`	`CHR(2)`	`CHR(0)`
	EBCDIC	`CHR(1)`	`CHR(255)`	Undefined

In the second element of the array, specify a number one less than the total number of characters in the collating sequence (*n* – 1). ASCII has 256 characters, for example, so for ASCII (the default), the second element would be CHR(255).

If the first element is CHR(0), the rest of the elements of the *`altseq`* parameter array contain your actual collating sequence. The collating sequence may be explicitly specified or read from a file. The sequence must start in the third element of the *`altseq`* array.

You can create a file that contains a collating sequence, using the SORT/XL or MERGE/XL utilities. For example, if you want to create the collating sequence mentioned above (AaBbCc ...) with the SORT/XL utility, you would do the following:

```
:FILE DISPLOUT=ALTSEQ,NEW; SAVE; REC=-80,,F,ASCII; DEV=DISC
:RUN SORT.PUB.SYS

>DATA IS ASCII, SEQUENCE IS ASCII
>ALTSEQ MERGE "A-Z" with "a-z"
>SHOW TABLE, OFFLINE
>EXIT
```

Now you have created an ASCII file named ALTSEQ that contains the new collating sequence in a table, a decimal list, and an octal list.

Edit the file to contain only the decimal list of the sequence. From your sorting or merging program, read in the sequence from the file ALTSEQ, and put the character representations of the values into elements 3-258 of the parameter *`altseq`*.

For more information about using SORT-MERGE/XL to create alternate collating sequences, refer to *SORT-MERGE/XL General User's Guide*.

Example of Using an Altered Sequence

The following example sorts the data file DATA. The entries in DATA are sorted using an altered collating sequence that is explicitly specified in the program. The sequence contains all displayable ASCII characters and alters the order of the alphabetic characters to AaBbCc ....The output file is called FRUIT.

DATA - a file of fruit names

```
banana
Apple
Grapes
grapes
Pear
peach
orange
```

Example 4-1. SORTALT Program

```
program SORTALT (input,output);

{This program  reads the file DATA, sorts by fruit name, }
{outputs to the file FRUIT, and uses an altered sequence.}

var
   dataFileNum : INTEGER;
   fruitFileNum: INTEGER;
   status      : INTEGER;

procedure HPFOPEN  ; intrinsic;
procedure HPSORTINIT; intrinsic;
procedure HPSORTEND; intrinsic;
procedure FCLOSE; intrinsic;
procedure OPEN_FILES;

const
   designator = 2;
   domain     = 3;
   access     = 11;
   record_Size = 19;

var
   datafile  : packed array [1..10] of CHAR;
   fruitfile : packed array [1..10] of CHAR;
   permanent : INTEGER;
   new       : INTEGER;
   write     : INTEGER;
   size      : INTEGER;
   begin
      datafile := '%DATA%';
      permanent := 1;
      HPFOPEN (dataFileNum, status, designator, datafile, domain, permanent)
;

      new := 4;
      write := 1;
      size := 80;
      fruitfile := '%FRUIT%';
      HPFOPEN (fruitFileNum, status, designator, fruitfile, domain, new,
access,
               write, record_size, size);

   end;

Procedure DO_SORT;

var
   inputfiles   : array [1..2] of INTEGER;
   outputfile   : array [1..2] of INTEGER;
   outputOption : INTEGER;
   numKeys      : INTEGER;
   keys         : array [1..4] of INTEGER;
   altseq       : packed array [1..96] of CHAR;

Begin
   inputfiles [1] := dataFileNum;
   inputfiles [2] := 0;

   outputfile [1] := fruitFileNum;                            {From HPFOPEN}
   outputfile [2] := 0;

   OutputOption := 0;      {Output record format same as input record format}

   numKeys := 1;                                                  {one key}
   keys[1] := 1;                                               {key begins}
   keys[2] := 20;                                              {key length}
   keys[3] := 0;                                                {byte data}
   keys[4] := 0;                                          {ascending order}

   altseq := ' ';
   altseq[1] := CHR(0);                    {Data = ASCII, Sequence is altered}
   altseq[2] := CHR(93);             {94 characters in altered ASCII sequence}

{Put sequence in Altseq.Sequence contains all displayable ASCII characters}

   strmove (15, ’!"#$%&’’)(*+,-./’, 1, altseq, 3);
   strmove (16, '0123456789::<=>?’, 1, altseq, 18);
   strmove (16, '@AaBbCcDdEeFfGgH', 1, altseq, 34);
   strmove (16, 'hIiJjKkLlMmNnOoP', 1, altseq, 50);
   strmove (16, 'pQqRrSsTtUuVvWwX', 1, altseq, 66);
   strmove (15, 'xYyZz[\][caret ]_{|}˜', 1, altseq, 81);
   HPSORTINIT (status, inputfiles, outputfile, outputOption,,,
      numKeys, keys, altseq,,,statistics,,);

   {Check for errors in HPSORTINIT}

   HPSORTEND (Status, );                    {Check for errors in HPSORTINIT}
end;

   Procedure CLOSE_FILES;

   var
      disposition : SHORTINT;
      securityCode : SHORTINT;

   Begin
      disposition := 0;
      securityCode := 0;
      FCLOSE (dataFileNum, disposition, securityCode);
      disposition := 1;
      FCLOSE (fruitFileNum, disposition, securityCode);

   end;

   begin {main}
      OPEN_FILES;
      DO_SORT;
      CLOSE_FILES;
   end.
```

When this program is executed, the output is written to FRUIT. To view FRUIT:

```
:print fruit

Apple
banana
Grapes
grapes
peach
Pear
orange
```

Using a Native Language Sequence

When you sort or merge information that has language considerations, you do so by using a collating sequence for Native Language types. These collating sequences include language-specific alphanumeric characters and accent marks.

Parameters in the HPSORTINIT or HPMERGEINIT intrinsic are used when specifying an alternate collating sequence. The parameters are:

- *keys* - Data type element must be set to 10 for character data.
- *charseq* - a 32-bit integer array. The first element contains the value of 1 (one) and second element contains the language ID of the collating sequence.

NOTE Use the *altseq* parameter with ASCII sequences only. Do not use it to create alternate sequences for native language (NL).

Getting the Language ID

To determine the supported languages and their language identification numbers (IDs), use the NLUTIL utility. For example, to sort or merge a German file, run NLUTIL to find the language ID for German:

Figure 4-1. Running NLUTIL

```
:RUN NLUTIL.PUB.SYS

HP32414A.03.00  NLUTIL/3000  (C) HEWLETT-PACKARD CO., 1983

   Lang   Lang                 Char    Char
     ID   Name                   ID    Name
   ----   ----                 ----    ----------------
      0   NATIVE-3000             0    USASCII
      1   AMERICAN                1    ROMAN8
      2   CANADIAN-FRENCH         1    ROMAN8
      3   DANISH                  1    ROMAN8
      4   DUTCH                   1    ROMAN8
      5   ENGLISH                 1    ROMAN8
      6   FINNISH                 1    ROMAN8
      7   FRENCH                  1    ROMAN8
      8   GERMAN                  1    ROMAN8
      9   ITALIAN                 1    ROMAN8
     10   NORWEGIAN               1    ROMAN8
     11   PORTUGUESE              1    ROMAN8
     12   SPANISH                 1    ROMAN8
     13   SWEDISH                 1    ROMAN8

Do you require a listing of the current configuration? N
Do you require a listing of one language? N
END OF PROGRAM
```

Specifying Parameters

You then specify the *keys* and *charseq* parameters for the HPSORTINIT or HPMERGEINIT intrinsic as follows:

```
var
  keys    : array [1..4] of INTEGER;
  charseq : array [1..2] of INTEGER;

  keys[1] := 1;  {key begins}
  keys[2] := 20; {key length}
  keys[3] := 10; {native language character data}
  keys[4] := 1;  {ascending order}

  charseq [1] := 1;
  charseq [1] := 8;  {language ID for GERMAN}
```

For detailed information about native language sorting and merging, refer to the *Native Language Programmer's Guide*.

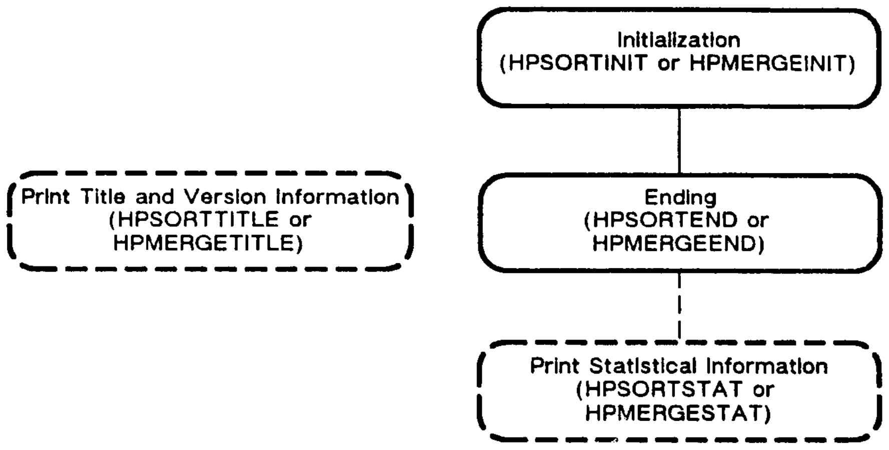
5 Getting SORT-MERGE/XL Information

SORT-MERGE/XL provides statistical information about the sorting or merging operation just completed, and title and version information about SORT-MERGE/XL.

The HPSORTSTAT and HPMERGESTAT intrinsics display status information about their processes. The HPSORTTITLE and HPMERGETITLE intrinsics display title and version information for their processes.

As shown in Figure 5-1. you may get title and version information at any point in your program. Statistical information is retreived after you have finished sorting or merging.

Figure 5-1. Getting Information



Redirecting Output

When you report statistics, title and version information, or error messages from SORT-MERGE/XL, this information is displayed to $STDLIST.

If you want this information in a file instead, you must redirect the output. To do this, build a file and tell MPE XL to put all the output meant for $STDLIST into the file. For example, to build a file named OUTFILE, use the BUILD command:

```
:BUILD OUTFILE;REC=-80,,,ASCII;DEV=DISC
```

To redirect output from the program SORTPROG to the file OUTFILE, specify at run time:

```
:RUN SORTPROG;STDLIST = OUTFILE

:END OF PROGRAM
```

You may also redirect $STDLIST to a back-referenced file or to $NULL.

Getting Statistical Information

For comparison or other purposes, SORT-MERGE/XL allows you to report statistical information. The statistical information provided by SORT-MERGE/XL includes:

- Number of input files (MERGE/XL only)
- Number of records sorted (or merged)
- Number of intermediate passes (SORT/XL only)
- Number of bytes used for the sort work area
- Number of comparisons (SORT/XL only)
- Amount of CPU time used (in milliseconds)
- Elapsed time (in milliseconds)

NOTE If SORT-MERGE/XL aborts, the values returned in *statistics* and printed out by HPSORTSTAT and HPMERGESTAT are unpredictable.

Syntax for HPSORTSTAT and HPMERGESTAT is:

```
HPSORTSTAT (status, statistics);

HPMERGESTAT (status, statistics);
```

HPSORTSTAT and HPMERGESTAT print the *statistics* array to $STDLIST.

An example of using HPSORTSTAT follows:

```
var
  status : INTEGER;
  statistics : array [1..6] of INTEGER;
     .
     .
     .
HPSORTSTAT (status, statistics);
```

The intrinsics and parameters required to get statistical information are:

- HPSORTINIT or HPMERGEINIT - *statistics* parameter must be specified.
- HPSORTEND or HPMERGEEND - *statistics* parameter must be specified.

HPSORTSTAT or HPMERGESTAT must be called *after* HPSORTEND or HPMERGEEND.

Example of Getting Sort/Merge Statistics

Using the *statistics* parameter in the HPSORTINIT and HPSORTEND intrinsics, and calling HPSORTSTAT from the core sorting routine example in Figure 2-1. on page 18 results in the following output:

```
**************************************************
*****************   STATISTICS *******************
**************************************************
 Number of records sorted =                       8
 Number of intermediate passes =                  0
 Number of bytes used for the sort work area =    27831
 Number of comparisons =                          6
 CPU time used (in millisecond) =                 45
 Elapsed time (in millisecond) =                  241
```

Using the *statistics* parameter in the HPMERGEINIT and HPMERGEEND intrinsics, and calling HPMERGESTAT from the core merging routine example in Chapter 2, results in the following output:

```
**************************************************
*****************   STATISTICS *******************
**************************************************
 Number of inputfiles =                           2
 Number of merged records =                       8
 Number of bytes used for the sort work area =    27794
 CPU time used (in millisecond) =                 46
 Elapsed time (in millisecond) =                  843
```

Getting Version and Title Information

You can print the version of SORT-MERGE/XL, the title of the HPSORTLIB or HPMERGELIB segment, the date, and the time to $STDLIST.

Syntax for HPSORTTITLE and HPMERGETITLE is:

```
HPSORTTITLE (status);

HPMERGETITLE (status);
```

The HPSORTTITLE or HPMERGETITLE intrinsic may be called anywhere in your program after you have declared the system intrinsics.

An example of using HPSORTTITLE follows:

```
var
  status : INTEGER;
     .
     .
     .
HPSORTTITLE (status);
```

Example of Using HPSORTTITLE

Calling HPSORTTITLE from the core sorting routine results in the following output:

```
HP31900A.01.00  SORT/XL THU, JAN 26, 1989, 10:28 AM
(C) HEWLETT-PACKARD CO. 1986
```

Calling HPMERGETITLE from the core merging routine results in the following output:

```
HP31900A.01.00  MERGE/XL THU, JAN 26, 1989, 10:28 AM
(C) HEWLETT-PACKARD CO. 1984
```

A HPSORT Intrinsic Status Returns

The following table lists the error number, message, cause, and user corrective action for status returns from the HPSORT intrinsics. SORT/XL's subsystem ID is 195.

`-1`	**MESSAGE**	`If you specified the` *`keycompare`* `parameter, you can not specify the` *`keys`* `and` *`numkeys`* `parameters.`
	CAUSE	You specified the *keycompare* and the *keys* and *numkeys* parameters in the `HPSORTINIT` intrinsic.
	ACTION	Specify the *keycompare* parameter or else specify the *keys* and *numkeys* parameters in the `HPSORTINIT`.
`-2`	**MESSAGE**	`If you do not specify the keycompare parameter, you must specify the keys and numkeys parameters.`
	CAUSE	You specified neither the *keycompare* parameter or the *keys* and *numkeys* parameters in the `HPSORTINIT` intrinsic.
	ACTION	Specify either the *keycompare* parameter or specify the *keys* and *numkeys* parameters in the `HPSORTINIT` intrinsic.
`-3`	**MESSAGE**	`You did not specify a reclength parameter, or you specified it less than or equal to 0.`
	CAUSE	You specified the *reclength* parameter in the `HPSORTINIT` intrinsic to be less than or equal to 0. or you did not specify an inputfile or use `HPSORTINPUT`.
	ACTION	Set the *reclength* parameter to a value that is greater than 1 or specify the *inputfiles* parameter use the `HPSORTINPUT` intrinsic.
`-4`	**MESSAGE**	`You must not specify the keycompare parameter if you specified the outputoption parameter as greater than 1.`
	CAUSE	The *keycompare* is only allowed when the entire record is output.
	ACTION	Change *outputoption* to 1 or don't use *keycompare*.

-5	**MESSAGE**	FREAD error on the scratch file.
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-6	**MESSAGE**	Illegal output option.
	CAUSE	You specified the *outputoption* parameter in HPSORTINIT to be less than zero or greater than three.
	ACTION	Set the *outputoption* parameter to 0, 1, 2, or 3.
-7	**MESSAGE**	The scratch file cannot be opened.
	CAUSE	A common cause of this problem is that there is not enough contiguous disk space available for the scratch file.
	ACTION	Compare the size of the file to be sorted with the available contiguous disk space. Purge or archive files to tape as necessary.
-8	**MESSAGE**	Failure on FGETINFO (inputfile).
	CAUSE	CCL condition code returned from the FGETINFO intrinsic called from SORT/XL.
	ACTION	Make sure the file number has not been corrupted. Also refer to the FGETINFO intrinsic in the *MPE XL Intrinsics Reference Manual*
-9	**MESSAGE**	Illegal numkeys parameter.
	CAUSE	You specified the *numkeys* parameter in HPSORTINIT to not correlate to the number of keys in the *keys* parameter.
	ACTION	Set the *numkeys* parameter to correspond to the number of keys that you specified in the *keys* parameter.
-10	**MESSAGE**	The key field is not within the specified record length.
	CAUSE	A key specified in your *keys* parameter in HPSORTINIT, indicates a field that begins and/or ends outside of the record.
	ACTION	Set the key field in the *keys* parameter to be contained in the record length or make sure that your record length is long enough to contain all your keys.
-11	**MESSAGE**	Illegal ascending/descending code.
	CAUSE	The 4th element for a key in the *keys* array parameter in HPSORTINIT is less than 0 or greater than 1.
	ACTION	Set the 4th element for a key in the *keys* array parameter to 0 or 1.

-12	**MESSAGE**	`Illegal key code.`
	CAUSE	The 3rd element in the *`keys`* array parameter in `HPSORTINIT` is less than 0 or greater than 13.
	ACTION	Set the 3rd element in the *`keys`* array parameter to a value that is from 0-13 inclusive.
-13	**MESSAGE**	`Insufficient stack space.`
	CAUSE	Input file was opened with `NOBUFF` and `MULTI` options. The stack was used for blocking/deblocking the file and has insufficient space.
	ACTION	Open your files with the `BUFF` and `NOMULTI` options.
-14	**MESSAGE**	`The input record does not include all key fields.`
	CAUSE	You are using variable length records that contain records that are not long enough to contain the key fields.
	ACTION	Set the key field in the *`keys`* parameter to be contained in the shortest record length or make sure that your shortest record is long enough to contain all your keys.
-15	**MESSAGE**	`The input record is too long.`
	CAUSE	The input record is longer than specified in the *`reclength`* parameter.
	ACTION	Alter the *`reclength`* parameter to accurately reflect the record length.
-16	**MESSAGE**	`There are too many input records.`
	CAUSE	SORT/XL internal error.
	ACTION	Contact your Hewlett-Packard representative.
-17	**MESSAGE**	`FWRITE error on the scratch file.`
	CAUSE	SORT/XL internal error.
	ACTION	Contact your Hewlett-Packard representative.
-18	**MESSAGE**	`FREAD error on the input file.`
	CAUSE	CCL returned from the `FREAD` intrinsic called from SORT/XL.
	ACTION	Make sure your input file number has not been corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.

-19	**MESSAGE**	`FWRITE error on the output file.`
	CAUSE	CCL returned from the `FWRITE` intrinsic called from SORT/XL.
	ACTION	Make sure your output file number has not been corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-20	**MESSAGE**	`FCLOSE error on the scratch file.`
	CAUSE	SORT/XL internal error.
	ACTION	Contact your Hewlett-Packard representative.
-21	**MESSAGE**	`$NULL is not a valid input file.`
	CAUSE	You specified `$NULL` as an input file.
	ACTION	Change the file specification to another file.
-22	**MESSAGE**	`Failure on FGETINFO (outputfile).`
	CAUSE	CCL returned from the `FGETINFO` intrinsic called from SORT/XL.
	ACTION	Make sure the output file number has not been corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-23	**MESSAGE**	`Error in attempting to write EOF on the scratch file.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-24	**MESSAGE**	`Error attempting to rewind the scratch file.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-25	**MESSAGE**	`Illegal characteristics for opening the scratch file with FOPEN.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-26	**MESSAGE**	`Insufficient stack space for the specified allocation.`
	CAUSE	The amount of space that you specified in the *`memsize`* parameter does not allow enough room to perform the sort.
	ACTION	Increase the value of the parameter or do not specify the *`memsize`*| parameter and let SORT/XL use as much stack space as it needs.

-27	**MESSAGE**	`Failure of FFILEINFO (inputfile).`
	CAUSE	CCL returned from the `FFILEINFO` intrinsic called from SORT/XL.
	ACTION	Make sure that the input file number is not corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-28	**MESSAGE**	`Failure of FFILEINFO (outputfile).`
	CAUSE	CCL returned from the `FFILEINFO` intrinsic called from SORT/XL.
	ACTION	Make sure that the output file number is not corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-29	**MESSAGE**	`The sort language is not supported.`
	CAUSE	The language specified in the second element of the *charseq* parameter array in `HPSORTINIT` is not supported on your system.
	ACTION	Check valid language IDs by running `NLUTIL`. Set the second element of the *charseq* array to a valid language ID.
-30	**MESSAGE**	`NLINFO error obtaining the length of the collating sequence table.`
	CAUSE	Native Language (NL) not installed or internal NL error.
	ACTION	Contact your system manager or your Hewlett-Packard representative.
-31	**MESSAGE**	`NLINFO error loading the collating sequence table.`
	CAUSE	Native Language (NL) not installed or internal NL error.
	ACTION	Contact your system manager or your Hewlett-Packard representative.
-32	**MESSAGE**	`Invalid charseq parameter.`
	CAUSE	The first element of the *charseq* parameter array is a value other than one (1).
	ACTION	Set the first element in the *charseq* parameter array to one.
-33	**MESSAGE**	`A two-byte collating sequence table is not specified.`
	CAUSE	A two-byte collating sequence table was not specified in the *charseq* parameter although a two-byte key was specified in the *keys* parameter.
	ACTION	Specify a two-byte sequence in both the *charseq* and *keys* parameters.

-34	**MESSAGE**	`FGETINFO failure on the two-byte collating sequence table.`
	CAUSE	CCL returned from the `FGETINFO` intrinsic called from SORT/XL.
	ACTION	Make sure the file number passed in *`charseq`* is not corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-35	**MESSAGE**	`FREAD error on the two-byte collating sequence table.`
	CAUSE	CCL returned from the `FREAD` intrinsic called from SORT/XL.
	ACTION	Make sure the file number passed in *`charseq`* is not corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-36	**MESSAGE**	`The file is not a valid two-byte collating sequence.`
	CAUSE	Error returned from `NLINFO` intrinsic called from SORT/XL.
	ACTION	Check the file type. Also refer to the *MPE XL Intrinsics Reference Manual*.
-37	**MESSAGE**	`Two-byte xxxx is undefined in the collating sequence table; the largest number is assigned.`
	CAUSE	The two-byte value, xxxx, is undefined.
	ACTION	Change xxxx to the correct value.
-38	**MESSAGE**	`The length of the two-byte key must be an even number of bytes.`
	CAUSE	Although you specified a two-byte sequence, the key length specified in the *`keys`* parameter is an odd number.
	ACTION	Specify the key length as an even number.
-39	**MESSAGE**	`The file type is not a valid two-byte collating sequence table.`
	CAUSE	The file containing the two-byte sequence has been corrupted, or Native Language (NL) not installed or internal NL error.
	ACTION	Contact your system manager or your Hewlett-Packard representative.

-40	**MESSAGE**	`PRINT intrinsic failed in HPSORTTITLE.`
	CAUSE	You redirected output from `HPSORTTITLE` to a file that is too small; `HPSORTTITLE` requires two records to output information.
	ACTION	Allow room for at least two records when you redirect output from `HPSORTTITLE` to a file.
-41	**MESSAGE**	`PRINT intrinsic failed in HPSORTSTAT.`
	CAUSE	You redirected output from `HPSORTSTAT` to a file that is too small; `HPSORTSTAT` requires ten records to output information.
	ACTION	Allow room for at least ten records when you redirect output from `HPSORTSTAT` to a file.
-190	**MESSAGE**	`There are too many input files.`
	CAUSE	You input more than 100 input files to SORT/XL.
	ACTION	Run SORT/XL several times with allowable number of files then run MERGE/XL to merge the sorted files or consolidate several files into one until you are within SORT/XL's limit.
-191	**MESSAGE**	`There are no input files in the supplied parameter.`
	CAUSE	You did not specify any input file numbers in the *`inputfiles`* parameter of the `HPSORTINIT` intrinsic. The array contains no file numbers.
	ACTION	Specify input file numbers in the *`inputfiles`* parameter of the `HPSORTINPUT` intrinsic.
-193	**MESSAGE**	`If you specify the keys parameter, you must also specify the numkeys parameter.`
	CAUSE	You specified the *`keys`* parameter without specifying the *`numkeys`* parameter in `HPSORTINIT` (or vice-versa).
	ACTION	Specify both the *`numkeys`* and the *`keys`* parameters.
-199	**MESSAGE**	`The record length exceeds the maximum allowed.`
	CAUSE	Internal File System error.
	ACTION	Contact your Hewlett-Packard representative.

-200	**MESSAGE**	`Insufficient memory allocated for the record size.`
	CAUSE	You are trying to sort more data than SORT/XL can handle.
	ACTION	If you are sorting one large file, break it into several smaller files. If you are sorting many large files, sort them individually and then merge them with MERGE/XL.
-201	**MESSAGE**	`Open of storage area failed.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-202	**MESSAGE**	`NM to CM switch occurred, but NM cannot handle the sort.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-203	**MESSAGE**	`Invalid input file number.`
	CAUSE	Input file number specified in *`inputfiles`* parameter of `HPSORTINIT` intrinsic is not a valid file number.
	ACTION	Don't hardcode a file number for the *`inputfiles`* parameter, make sure the file number returned from the `HPFOPEN` or `FOPEN` intrinsic is not corrupted.
-204	**MESSAGE**	`Record length exceeds maximum allowed.`
	CAUSE	Internal File System error.
	ACTION	Contact your Hewlett-Packard representative.
-250	**MESSAGE**	`PROBE failure on the status parameter of the HPSORTINIT intrinsic.`
	CAUSE	The address specified in the *`status`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`status`* parameter.
-251	**MESSAGE**	`PROBE failure on the inputfiles parameter of the HPSORTINIT intrinsic.`
	CAUSE	The address specified in the *`inputfiles`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`inputfiles`* parameter.
-252	**MESSAGE**	`PROBE failure on the outputfiles parameter of the HPSORTINIT intrinsic.`
	CAUSE	The address specified in the *`outputfiles`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`outputfiles`* parameter.

-253	**MESSAGE**	`PROBE failure on the keys parameter of the HPSORTINIT intrinsic.`
	CAUSE	The address specified in the *keys* parameter is not within the allowable address range.
	ACTION	Check the value of the *keys* parameter.
-254	**MESSAGE**	`PROBE failure on the altseq parameter of the HPSORTINIT intrinsic.`
	CAUSE	The address specified in the *altseq* parameter is not within the allowable address range.
	ACTION	Check the value of the *altseq* parameter.
-255	**MESSAGE**	`PROBE failure on the statistics parameter of the HPSORTINIT intrinsic.`
	CAUSE	The address specified in the *statistics* parameter is not within the allowable address range.
	ACTION	Check the value of the *statistics* parameter.
-256	**MESSAGE**	`PROBE failure on the charseq parameter of the HPSORTINIT intrinsic.`
	CAUSE	The address specified in the *charseq* parameter is not within the allowable address range.
	ACTION	Check the value of the *charseq* parameter.
-257	**MESSAGE**	`PROBE failure on the status parameter of the HPSORTINPUT intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
-258	**MESSAGE**	`PROBE failure on the buffer parameter of the HPSORTINPUT intrinsic.`
	CAUSE	The address specified in the *buffer* parameter is not within the allowable address range.
	ACTION	Check the value of the *buffer* parameter.
-259	**MESSAGE**	`PROBE failure on the length parameter of the HPSORTINPUT intrinsic.`
	CAUSE	The address specified in the *length* parameter is not within the allowable address range.
	ACTION	Check the value of the *length* parameter.

`-260`	**MESSAGE**	`PROBE failure on the status parameter of the HPSORTOUTPUT intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
`-261`	**MESSAGE**	`PROBE failure on the buffer parameter of the HPSORTOUTPUT intrinsic.`
	CAUSE	The address specified in the *buffer* parameter is not within the allowable address range.
	ACTION	Check the value of the *buffer* parameter.
`-262`	**MESSAGE**	`PROBE failure on the length parameter of the HPSORTOUTPUT intrinsic.`
	CAUSE	The address specified in the *length* parameter is not within the allowable address range.
	ACTION	Check the value of the *length* parameter.
`-263`	**MESSAGE**	`PROBE failure on the status parameter of the HPSORTEND intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
`-264`	**MESSAGE**	`PROBE failure on the statistics parameter of the HPSORTEND intrinsic.`
	CAUSE	The address specified in the *statistics* parameter is not within the allowable address range.
	ACTION	Check the value of the *statistics* parameter.
`-265`	**MESSAGE**	`PROBE failure on the status parameter of the HPSORTERRORMESS intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
`-266`	**MESSAGE**	`PROBE failure on the message parameter of the HPSORTERRORMESS intrinsic.`
	CAUSE	The address specified in the *message* parameter is not within the allowable address range.
	ACTION	Check the value of the *message* parameter.

-267	**MESSAGE**	`PROBE failure on the length parameter of the HPSORTERRORMESS intrinsic.`
	CAUSE	The address specified in the *`length`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`length`* parameter.
-268	**MESSAGE**	`PROBE failure on the status parameter of the HPSORTSTAT intrinsic.`
	CAUSE	The address specified in the *`status`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`status`* parameter.
-269	**MESSAGE**	`PROBE failure on the statistics parameter of the HPSORTSTAT intrinsic.`
	CAUSE	The address specified in the *`statistics`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`statistics`* parameter.
-270	**MESSAGE**	`PROBE failure on the status parameter of the HPSORTTITLE intrinsic.`
	CAUSE	The address specified in the *`status`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`status`* parameter.
-990	**MESSAGE**	`Previous NM error occurred.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-992	**MESSAGE**	`SWITCH_TO_CM error on the SORTTITLE call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-993	**MESSAGE**	`SWITCH_TO_CM error on the SORTERRORMESS call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-994	**MESSAGE**	`SWITCH_TO_CM error on the SORTEND2 call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-995	**MESSAGE**	`SWITCH_TO_CM error on the SORTEND1 call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.

`-996`	**MESSAGE**	`SWITCH_TO_CM error on the SORTOUTPUT call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
`-997`	**MESSAGE**	`SWITCH_TO_CM error on the SORTINPUT call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
`-998`	**MESSAGE**	`SWITCH_TO_CM error on the SORTGETHIDP call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
`-999`	**MESSAGE**	`SWITCH_TO_CM error on the SORTINIT call.`
	CAUSE	Internal SORT/XL error.
	ACTION	Contact your Hewlett-Packard representative.
`-1000`	**MESSAGE**	`HPSORTERRORMESS failed on the call to HPERRMSG.`
	CAUSE	You called the HPSORTERRORMESS intrinsic even though there was no error.
	ACTION	Only call HPSORTERRORMESS if the *status* parameter from the previous intrinsic call does not equal zero.

B HPMERGE Intrinsic Status Returns

The following table lists the error number, message, cause, and user corrective action for status returns for all HPMERGE intrinsics. MERGE/XL's subsystem ID is 196.

-3	**MESSAGE**	`No inputfiles parameter was specified.`
	CAUSE	You did not specify the *inputfiles* parameter in the `HPMERGEINIT` intrinsic.
	ACTION	Specify the *inputfiles* parameter.
-4	**MESSAGE**	`Neither an outputfiles nor a postprocessor parameter was specified.`
	CAUSE	You did not specify the *outputfiles* parameter in the `HPMERGEINIT` intrinsic.
	ACTION	Specify the *outputfiles* parameter.
-5	**MESSAGE**	`If you specify the keycompare parameter, you must not specify the keys and numkeys parameters.`
	CAUSE	You specified both the *keycompare* parameter, and the *keys* and *numkeys* parameters in the `HPMERGEINIT` intrinsic.
	ACTION	Specify the *keycompare* parameter or the *keys* and *numkeys* parameters.
-6	**MESSAGE**	`If you do not specify the keycompare parameter, you must specify the keys and numkeys parameters.`
	CAUSE	You did not specify the *keys* and *numkeys* parameters in the `HPMERGEINIT` intrinsic.
	ACTION	Specify the *keys* and *numkeys* parameters.
-7	**MESSAGE**	`Illegal numkeys parameter.`
	CAUSE	The *numkeys* parameter does not specify the same number of keys that are described in the the *numkeys* parameter in `HPMERGEINIT`.
	ACTION	Make the *numkeys* and *keys* parameters pertain to the same number of keys.

-8	**MESSAGE**	`The key field is not within the record length of each file.`
	CAUSE	One or more files have shorter record lengths and at least one key field extends outside of the file's record length.
	ACTION	Make sure your files are of the correct length and that your key field in within range of the records.
-9	**MESSAGE**	`Illegal ascending/descending code.`
	CAUSE	The fourth element of the *keys* array parameter is not 1 or 0.
	ACTION	Set the fourth element of the *keys* array parameter to 0 for ascending or 1 for descending order.
-10	**MESSAGE**	`Illegal key code.`
	CAUSE	The 3rd element in the *keys* array parameter in `HPMERGEINIT` is less than 0 or greater than 13.
	ACTION	Set the 3rd element in the *keys* array parameter to a value that is from 0-13 inclusive.
-11	**MESSAGE**	`Failure on FGETINFO (inputfile).`
	CAUSE	CCL was returned from the `FGETINFO` intrinsic called from SORT/XL.
	ACTION	Make sure that the input file number hasn't been corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-12	**MESSAGE**	`FREAD error on the input file.`
	CAUSE	CCL was returned from the `FREAD` intrinsic called from SORT/XL.
	ACTION	Make sure that the input file number hasn't been corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-13	**MESSAGE**	`FWRITE error on the output file.`
	CAUSE	CCL was returned from the `FWRITE` intrinsic called from SORT/XL.
	ACTION	Make sure that the output file number hasn't been corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.

-14	**MESSAGE**	`The input record does not include all key fields.`
	CAUSE	You are using variable length records that contain records that not long enough to contain the key fields.
	ACTION	Set the key field in the *keys* parameter to be contained in the shortest record length or make sure that your shortest record is long enough to contain all your keys.
-15	**MESSAGE**	`If you specify the keycompare parameter, you must not specify the keysonly parameter.`
	CAUSE	You specified the *keycompare* parameter and the *keys* and *numkeys* parameters in the HPMERGEINIT intrinsic.
	ACTION	Specify either the *keycompare* parameter or the *keys* and *numkeys* parameters in HPMERGEINIT.
-16	**MESSAGE**	`Insufficient stack space.`
	CAUSE	File was opened with NOBUFF and MULTI options. The stack was used for blocking/deblocking the file and has insufficient space.
	ACTION	Open your files with the BUFF and NOMULTI options.
-17	**MESSAGE**	`Insufficient stack space for the specified allocation.`
	CAUSE	The amount of space that you specified in the *memsize* parameter does not allow enough room to perform the merging operation.
	ACTION	Increase the value of the *memsize* parameter or do not specify the *memsize* parameter and let MERGE/XL use as much stack space as it needs.
-18	**MESSAGE**	`Failure on FGETINFO (outputfile).`
	CAUSE	CCL returned from the *FGETINFO* intrinsic called from MERGE/XL.
	ACTION	Make sure the output file number has not been corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-19	**MESSAGE**	`$NULL is not a valid input file.`
	CAUSE	You specified $NULL as an input file.
	ACTION	Change the file specification to another file.

-21	**MESSAGE**	`Sort language is not supported.`
	CAUSE	The language specified in the second element of the *charseq* parameter array in `HPMERGEINIT` is not supported on your system.
	ACTION	Check valid language IDs by running `NLUTIL`. Set the second element of the *charseq* parameter array to a valid language ID.
-22	**MESSAGE**	`NLINFO error obtaining the length of the collating sequence table.`
	CAUSE	Native Language (NL) not installed or internal NL error.
	ACTION	Contact your system manager or your Hewlett-Packard representative.
-23	**MESSAGE**	`NLINFO error loading the collating sequence.`
	CAUSE	Native Language (NL) not installed or internal NL error.
	ACTION	Contact your system manager or your Hewlett-Packard representative.
-24	**MESSAGE**	`Invalid charseq parameter.`
	CAUSE	The first element of the *charseq* parameter array is a value other than one.
	ACTION	Set the first element in the *charseq* array to one.
-25	**MESSAGE**	`The two-byte collating sequence table is not specified.`
	CAUSE	Two-byte collating sequence table was not specified in the *charseq* parameter although a two-byte key was specified in the *keys* parameter.
	ACTION	Specify a two-byte sequence in both the *charseq* and *keys* parameters.
-26	**MESSAGE**	`Failure on FGETINFO (two-byte collating sequence table).`
	CAUSE	CCL returned from the `FGETINFO` intrinsic called from MERGE/XL.
	ACTION	Make sure the file number in *charseq* is not corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.
-27	**MESSAGE**	`FREAD error on the two-byte collating sequence table.`
	CAUSE	CCL returned from the `FREAD` intrinsic called from MERGE/XL.
	ACTION	Make sure the file number in *charseq* is not corrupted. Also refer to the *MPE XL Intrinsics Reference Manual*.

-28	**MESSAGE**	`The file is not a valid two-byte collating sequence table.`
	CAUSE	Error returned from `NLINFO` intrinsic called from MERGE/XL.
	ACTION	Check the file type. Also refer to the *MPE XL Intrinsics Reference Manual*.
-29	**MESSAGE**	`Two-byte xxxx undefined in the collating sequence table; the largest number is assigned.`
	CAUSE	The two-byte value, xxxx, is undefined.
	ACTION	Change xxxx to the correct value.
-30	**MESSAGE**	`The length of the two-byte key must be an even number.`
	CAUSE	The second element of the *keys* array parameter must be an even number if a two-byte collating sequence is specified.
	ACTION	Set the second element of the *keys* array to an even number.
-31	**MESSAGE**	`The file type is not a valid two-byte collating sequence table.`
	CAUSE	The file containing the two-byte sequence has been corrupted or Native Language (NL) not installed or internal NL error.
	ACTION	Contact your system manager or your Hewlett-Packard representative.
-40	**MESSAGE**	`PRINT intrinsic failed in HPMERGETITLE.`
	CAUSE	You redirected output from `HPMERGETITLE` to a file that is too small; `HPMERGETITLE` requires two records in which to output information.
	ACTION	Allow room for at lease two records when you redirect output from `HPMERGETITLE` to a file.
-41	**MESSAGE**	`PRINT intrinsic failed in HPMERGESTAT.`
	CAUSE	You redirected output from `HPMERGESTAT` to a file that is too small; `HPMERGESTAT` requires ten records in which to output information.
	ACTION	Allow room for at least ten records when you redirect output from `HPMERGESTAT` to a file.

`-109`	**MESSAGE**	`Illegal numkeys parameter.`
	CAUSE	You specified the *`numkeys`* parameter in `HPMERGEINIT` to not correlate to the number of keys in the *`keys`* parameter.
	ACTION	Set the *`numkeys`* parameter to correspond to the number of keys that you specified in the *`keys`* parameter.
`-250`	**MESSAGE**	`PROBE failure on the status parameter of the HPMERGEINIT intrinsic.`
	CAUSE	The address specified in the *`status`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`status`* parameter.
`-251`	**MESSAGE**	`PROBE failure on the inputfiles parameter of the HPMERGEINIT intrinsic.`
	CAUSE	The address specified in the *`inputfiles`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`inputfiles`* parameter.
`-252`	**MESSAGE**	`PROBE failure on the outputfiles parameter of the HPMERGEINIT intrinsic.`
	CAUSE	The address specified in the *`outputfiles`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`outputfiles`* parameter.
`-253`	**MESSAGE**	`PROBE failure on the keys parameter of the HPMERGEINIT intrinsic.`
	CAUSE	The address specified in the *`keys`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`keys`* parameter.
`-254`	**MESSAGE**	`PROBE failure on the altseq parameter of the HPMERGEINIT intrinsic.`
	CAUSE	The address specified in the *`altseq`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`altseq`* parameter.
`-255`	**MESSAGE**	`PROBE failure on the statistics parameter of the HPMERGEINIT intrinsic.`
	CAUSE	The address specified in the *`statistics`* parameter is not within the allowable address range.
	ACTION	Check the value of the *`statistics`* parameter.

-256	**MESSAGE**	`PROBE failure on the charseq parameter of the HPMERGEINIT intrinsic.`
	CAUSE	The address specified in the *charseq* parameter is not within the allowable address range.
	ACTION	Check the value of the *charseq* parameter.
-260	**MESSAGE**	`PROBE failure on the status parameter of the HPMERGEOUTPUT intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
-261	**MESSAGE**	`PROBE failure on the buffer parameter of the HPMERGEOUTPUT intrinsic.`
	CAUSE	The address specified in the *buffer* parameter is not within the allowable address range.
	ACTION	Check the value of the *buffer* parameter.
-262	**MESSAGE**	`PROBE failure on the length parameter of the HPMERGEOUTPUT intrinsic.`
	CAUSE	The address specified in the *length* parameter is not within the allowable address range.
	ACTION	Check the value of the *length* parameter.
-263	**MESSAGE**	`PROBE failure on the status parameter of the HPMERGEEND intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
-264	**MESSAGE**	`PROBE failure on the statistics parameter of the HPMERGEEND intrinsic.`
	CAUSE	The address specified in the *statistics* parameter is not within the allowable address range.
	ACTION	Check the value of the *statistics* parameter.
-265	**MESSAGE**	`PROBE failure on the status parameter of the HPMERGEERRORMESS intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.

-266	**MESSAGE**	`PROBE failure on the message parameter of the HPMERGEERRORMESS intrinsic.`
	CAUSE	The address specified in the *message* parameter is not within the allowable address range.
	ACTION	Check the value of the *message* parameter.
-267	**MESSAGE**	`PROBE failure on the length parameter of the HPMERGEERRORMESS intrinsic.`
	CAUSE	The address specified in the *length* parameter is not within the allowable address range.
	ACTION	Check the value of the *length* parameter.
-268	**MESSAGE**	`PROBE failure on the status parameter of the HPMERGESTAT intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
-269	**MESSAGE**	`PROBE failure on the statistics parameter of the HPMERGESTAT intrinsic.`
	CAUSE	The address specified in the *statistics* parameter is not within the allowable address range.
	ACTION	Check the value of the *statistics* parameter.
-270	**MESSAGE**	`PROBE failure on the status parameter of the HPMERGETITLE intrinsic.`
	CAUSE	The address specified in the *status* parameter is not within the allowable address range.
	ACTION	Check the value of the *status* parameter.
-993	**MESSAGE**	`SWITCH_TO_CM failed on the MERGETITLE call.`
	CAUSE	Internal MERGE/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-994	**MESSAGE**	`SWITCH_TO_CM failed on the MERGEERRORMESS call.`
	CAUSE	Internal MERGE/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-995	**MESSAGE**	`SWITCH_TO_CM failed on the MERGEEND2 call.`
	CAUSE	Internal MERGE/XL error.
	ACTION	Contact your Hewlett-Packard representative.

-996	**MESSAGE**	`SWITCH_TO_CM failed on the MERGEEND1 call.`
	CAUSE	Internal MERGE/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-997	**MESSAGE**	`SWITCH_TO_CM failed on the MERGEOUTPUT call.`
	CAUSE	Internal MERGE/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-998	**MESSAGE**	`SWITCH_TO_CM failed on the MERGEGETHIDP call.`
	CAUSE	Internal MERGE/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-999	**MESSAGE**	`SWITCH_TO_CM failed on the MERGEINIT call.`
	CAUSE	Internal MERGE/XL error.
	ACTION	Contact your Hewlett-Packard representative.
-1000	**MESSAGE**	`HPMERGEERRORMESS failed on the call to HPERRMSG.`
	CAUSE	You called the HPMERGEERRORMESS intrinsic even though there was no error.
	ACTION	Only call HPMERGEERRORMESS if the *status* parameter from the previous intrinsic call does not equal zero.

C ASCII/EBCDIC Table

The following table shows ASCII and EBCDIC character code values along with their decimal, octal, and hexadecimal equivalents. Abreviations, such as NUL and SOH are spelled out at the end of the table, in order of appearance.

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
NUL	NUL	0	000	00
SOH	SOH	1	001	01
STX	STX	2	002	02
ETX	ETX	3	003	03
EOT	PF	4	004	04
ENQ	HT	5	005	05
ACK	LC	6	006	06
BEL	DEL	7	007	07
BS		8	010	08
HT		9	011	09
LF	SMM	10	012	0A
VT	VT	11	013	0B
FF	FF	12	014	0C
CR	CR	13	015	0D
SO	SO	14	016	0E
SI	SI	15	017	0F
DLE	DLE	16	020	10
DC1	DC1	17	021	11
DC2	DC2	18	022	12
DC3	TM	19	023	13

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
DC4	RES	20	024	14
NAK	NL	21	025	15
SYN	BS	22	026	16
ETB	IL	23	027	17
CAN	CAN	24	030	18
EM	EM	25	031	19
SUB	CC	26	032	1A
ESC	CU1	27	033	1B
FS	IFS	28	034	1C
GS	IGS	29	035	1D
RS	IRS	30	036	1E
US	IUS	31	037	1F
SP	DS	32	040	20
!	SOS	33	041	21
"	FS	34	042	22
#		35	043	23
$	BYP	36	044	24
%	LF	37	045	25
&	ETB	38	046	26
'	ESC	39	047	27
(		40	050	28
)		41	051	29
*	SM	42	052	2A
+	CU2	43	053	2B
,		44	054	2C
-	ENQ	45	055	2D
.	ACK	46	056	2E
/	BEL	47	057	2F

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
0		48	060	30
1		49	061	31
2	SYN	50	062	32
3		51	063	33
4	PN	52	064	34
5	RS	53	065	35
6	UC	54	066	36
7	EOT	55	067	37
8		56	070	38
9		57	071	39
:		58	072	3A
;	CU3	59	073	3B
<	DC4	60	074	3C
=	NAK	61	075	3D
>		62	076	3E
?	SUB	63	077	3F
@	SP	64	100	40
A		65	101	41
B		66	102	42
C		67	103	43
D		68	104	44
E		69	105	45
F		70	106	46
G		71	107	47
H		72	110	48
I		73	111	49
J		74	112	4A
K	.	75	113	4B

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
L	<	76	114	4C
M	(	77	115	4D
N	+	78	116	4E
O	|	79	117	4F
P	&	80	120	50
Q		81	121	51
R		82	122	52
S		83	123	53
T		84	124	54
U		85	125	55
V		86	126	56
W		87	127	57
X		88	130	58
Y		89	131	59
Z	!	90	132	5A
[	$	91	133	5B
\	*	92	134	5C
]	)	93	135	5D
^	;	94	136	5E
_		95	137	5F
`	-	96	140	60
a	/	97	141	61
b		98	142	62
c		99	143	63
d		100	144	64
e		101	145	65
f		102	146	66
g		103	147	67

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
h		104	150	68
i		105	151	69
j		106	152	6A
k	,	107	153	6B
l	%	108	154	6C
m	_	109	155	6D
n	>	110	156	6E
o	?	111	157	6F
p		112	160	70
q		113	161	71
r		114	162	72
s		115	163	73
t		116	164	74
u		117	165	75
v		118	166	76
w		119	167	77
x		120	170	78
y		121	171	79
z	:	122	172	7A
{	#	123	173	7B
|	@	124	174	7C
}	'	125	175	7D
~	=	126	176	7E
DEL	"	127	177	7F
		128	200	80
	a	129	201	81
	b	130	202	82
	c	131	203	83

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
	d	132	204	84
	e	133	205	85
	f	134	206	86
	g	135	207	87
	h	136	210	88
	i	137	211	89
		138	212	8A
		139	213	8B
		140	214	8C
		141	215	8D
		142	216	8E
		143	217	8F
	j	144	220	90
	k	145	221	91
	l	146	222	92
		147	223	93
	m	148	224	94
	n	149	225	95
	o	150	226	96
	p	151	227	97
	q	152	230	98
	r	153	231	99
		154	232	9A
		155	233	9B
		156	234	9C
		157	235	9D
		158	236	9E
		159	237	9F

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
		160	240	A0
	~	161	241	A1
	s	162	242	A2
	t	163	243	A3
	u	164	244	A4
	v	165	245	A5
	w	166	246	A6
	x	167	247	A7
	y	168	250	A8
	z	169	251	A9
		170	252	AA
		171	253	AB
		172	254	AC
		173	255	AD
		174	256	AE
		175	257	AF
		176	260	B0
		177	261	B1
		178	262	B2
		179	263	B3
		180	264	B4
		181	265	B5
		182	266	B6
		183	267	B7
		184	270	B8
		185	271	B9
		186	272	BA
		187	273	BB

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
		188	274	BC
		189	275	BD
		190	276	BE
		191	277	BF
		192	300	C0
	A	193	301	C1
	B	194	302	C2
	C	195	303	C3
	D	196	304	C4
	E	197	305	C5
	F	198	306	C6
	G	199	307	C7
	H	200	310	C8
	I	201	311	C9
		202	312	CA
		203	313	CB
		204	314	CC
		205	315	CD
		206	316	CE
		207	317	CF
		208	320	D0
	J	209	321	D1
	K	210	322	D2
	L	211	323	D3
	M	212	324	D4
	N	213	325	D5
	O	214	326	D6
	P	215	327	D7

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
	Q	216	330	D8
	R	217	331	D9
		218	332	DA
		219	333	DB
		220	334	DC
		221	335	DD
		222	336	DE
		223	337	DF
	\	224	340	E0
		225	341	E1
	S	226	342	E2
	T	227	343	E3
	U	228	344	E4
	V	229	345	E5
	W	230	346	E6
	X	231	347	E7
	Y	232	350	E8
	Z	233	351	E9
		234	352	EA
		235	353	EB
		236	354	EC
		237	355	ED
		238	356	EE
		239	357	EF
	0	240	360	F0
	1	241	361	F1
	2	242	362	F2
	3	243	363	F3

Table C-1. ASCII/EBCDIC Character Sets

ASCII	EBCDIC	Character Code Values		
Control/Graphic	Control/Graphic	Decimal	Octal	Hexadecimal
	4	244	364	F4
	5	245	365	F5
	6	246	366	F6
	7	247	367	F7
	8	248	370	F8
	9	249	371	F9
		250	372	FA
		251	373	FB
		252	374	FC
		253	375	FD
		254	376	FE
		255	377	FF

`NUL`	Null
`SOH`	Start of Heading
`STX`	Start of Text
`ETX`	End of Text
`EOT`	End of Transmission
`ENQ`	Enquiry
`ACK`	acknowledge
`BEL`	Bell
`BS`	Backspace
`HT`	Horizontal Tabulation
`LF`	Line Feed
`VT`	Vertical Tabulation
`FF`	Form Feed
`CR`	Carriage Return
`SO`	Shift Out
`SI`	Shift In
`DLE`	Data Link Escape
`DC1`	Device Control 1 (X-ON)
`DC2`	Device Control 2
`DC3`	Device Control 3 (X-OFF)
`DC4`	Device Control 4
`NAK`	Negative Acknowledge
`SYN`	Synchronous Idle
`ETB`	End of Transmission Block
`CAN`	Cancel
`EM`	End of Medium
`SUB`	Substitute
`ESC`	Escape
`FS`	File Separator
`GS`	Group Separator
`RS`	Record Separator
`US`	Unit Separator
`SP`	Space (Blank)
`DEL`	Delete

D FORTRAN Program Examples

The following examples are included in this appendix (the location of the topic discussion is indicated in parentheses):

- Example of Core Sorting Routine (Chapter 2) - `SORTFILE`
- Example of Core Merging Routine (Chapter 2) - `MERGEFILE`
- Example of Record Input (Chapter 3) - `SORTREC_INPUT`
- Example of Record Output (Chapter 3) - `SORTREC_OUTPUT`
- Example of Using an Altered Sequence (Chapter 4) - `SORTALT`

Example of Core Sorting Routine

The following program sorts the personnel files shown below. They are sorted together by last name. The record size is determined by the input files. The *status* parameter is checked after the calls to HPSORTINIT and HPSORTEND.

The files that are used in this example are as follows (character positions and data descriptions are for convenience only):

TEMPEMP information file about temporary employees:

```
Last Name           First Name       Employee Number      Hire Date

Gangley,            Tomas               000003             06/06/87
Rields,             Evelyn              000007             07/12/87
Everett,            Joyce               000029             10/19/87

0        1         2         3         4         5         6         7

1234567890123456789012345678901234567890123456789012345678901234567890
```

PERMEMP information file about permanent employees:

```
Last Name          First Name        Employee Number      Hire Date

Jones,              Eliza               000001             06/06/87
Smith,              James               000005             06/06/87
Jackson,            Johnathon           000006             06/06/87
Washington,         Lois                000014             07/23/87
Jackson,            Rosa                000022             08/15/87

0        1         2         3         4         5         6         7

1234567890123456789012345678901234567890123456789012345678901234567890
```

Example D-1. SORTFILE Program

```
$standard_level system
      program SORTFILE
C
C     This program reads the files TEMPEMP and PERMEMP, sorts by last name,
C  and outputs to the file ALLEMP.  The compiler directive '$standard_level
C  system' is used to supress FORTRAN 77 warnings for non-standard features,
C  which include intrinsics calls.
C
      integer TEMPFILENUM
     2       ,PERMFILENUM
     3       ,OUTFILENUM
     4       ,STATUS
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      call OPEN_FILES
      call DO_SORT
      call CLOSE_FILES
      stop
      end
C
      subroutine OPEN_FILES
C
      system intrinsic HPFOPEN
     2                ,QUIT
C
      integer DESIGNATOR
     2       ,DOMAIN
     3       ,ACCESS
     4       ,RECORD_SIZE
     5       ,PERMANENT
     6       ,NEW
     7       ,WRITE
     8       ,SIZE
     9       ,TEMPFILENUM
     A       ,PERMFILENUM
     B       ,OUTFILENUM
     C       ,STATUS
C
      character TEMPFILE*10
     2         ,PERMFILE*10
     3         ,OUTFILE*10
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      DESIGNATOR  = 2
      DOMAIN      = 3
      ACCESS      = 11
      RECORD_SIZE = 19
C
      TEMPFILE = '%TEMPEMP%'
      PERMANENT = 1
      call HPFOPEN (TEMPFILENUM, STATUS, DESIGNATOR,
     2             ,TEMPFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
      PRINT *,STATUS
        print *,'HPFOPEN error on TEMPEMP.  Terminating.'
        call QUIT (1)
      endif
C
      PERMFILE = '%PERMEMP%'
      call HPFOPEN (PERMFILENUM, STATUS, DESIGNATOR,
     2             ,PERMFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
        print *,'HPFOPEN error on PERMEMP.  Terminating.'
        call QUIT (2)
      endif
C
      NEW   = 4
      WRITE = 1
      SIZE  = 80

      OUTFILE = '%ALLEMP%'
      call HPFOPEN (OUTFILENUM, STATUS, DESIGNATOR,
     2             ,OUTFILE, DOMAIN, NEW, ACCESS, WRITE
     3             ,RECORD_SIZE, SIZE)
      if (STATUS .ne. 0) then
        print *,'HPFOPEN error on ALLEMP.  Terminating.'
      endif
C
      return
      end
C
      subroutine DO_SORT
C
      system intrinsic HPSORTINIT
     2                ,HPSORTERRORMESS
     3                ,HPSORTEND
C
      integer OUTPUT_OPTION
     2       ,NUMKEYS
     3       ,LENGTH
     4       ,INPUTFILES(3)
     5       ,OUTPUTFILE(2)
     6       ,KEYS(4)
     7       ,STATISTICS(6)
     8       ,TEMPFILENUM
     9       ,PERMFILENUM
     A       ,OUTFILENUM
     B       ,STATUS
C
      character ALTSEQ*2
     2         ,MESSAGE*80
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      INPUTFILES(1) = TEMPFILENUM
      INPUTFILES(2) = PERMFILENUM
      INPUTFILES(3) = 0
      OUTPUTFILE(1) = OUTFILENUM
      OUTPUTFILE(2) = 0

      OUTPUT_OPTION = 0
      NUMKEYS       = 1
      KEYS(1)       = 1
      KEYS(2)       = 20
      KEYS(3)       = 0
      KEYS(4)       = 0

      ALTSEQ(1:1)   = CHAR(255)
      ALTSEQ(1:2)   = CHAR(255)

      call HPSORTINIT (STATUS, INPUTFILES, OUTPUTFILE,
     2      OUTPUT_OPTION,,, NUMKEYS, KEYS, ALTSEQ)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif

      call HPSORTEND (STATUS,STATISTICS)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif

      return
      end
C
      subroutine CLOSE_FILES
C
      system intrinsic FCLOSE
C
      integer*2 DISPOSITION
     2         ,SECURITYCODE
C
      integer TEMPFILENUM
     2       ,PERMFILENUM
     3       ,OUTFILENUM
     4       ,STATUS
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      DISPOSITION = 0
      SECURITYCODE = 0
C
      call FCLOSE (TEMPFILENUM, DISPOSITION, SECURITYCODE)
      call FCLOSE (PERMFILENUM, DISPOSITION, SECURITYCODE)
      DISPOSITION = 1
      call FCLOSE (OUTFILENUM, DISPOSITION, SECURITYCODE)
      return
      end
```

When this program is executed, the output from the sort is written to ALLEMP. To view the output:

```
:print allemp

Everett,           Joyce              000029             10/19/87
Gangley,           Tomas              000003             06/06/87
Jackson,           Jonathan           000006             06/06/87
Jackson,           Rosa               000022             08/15/87
Jones,             Eliza              000001             06/06/87
Rields,            Evelyn             000007             07/12/87
Smith,             James              000005             06/06/87
Washington,        Lois               000014             07/23/87
```

Example of Core Merging Routine

The following program merges the personnel files shown at the beginning of the previous example. They are merged by employee number. The record size is determined by the input files. The *status* parameter is checked after the calls to HPMERGEINIT and HPMERGEEND.

Example D-2. MERGEFILE Program

```
$standard_level system
      program MERGEFILE
C
C     This program reads the files TEMPEMP and PERMEMP, merges them by EMPLOYEE
C  NUMBER, and outputs them to the file ALLEMP.
C  The compiler directive '$standard_level system' is used to supress
C  FORTRAN 77 warnings for non-standard features, which include intrinsics
C  calls.
C
      integer TEMPFILENUM
     2       ,PERMFILENUM
     3       ,OUTFILENUM
     4       ,STATUS
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      call OPEN_FILES
      call DO_MERGE
      call CLOSE_FILES
      stop
      end
C
      subroutine OPEN_FILES
C
      system intrinsic HPFOPEN
     2                ,QUIT
C
      integer DESIGNATOR
     2       ,DOMAIN
     3       ,ACCESS
     4       ,RECORD_SIZE
     5       ,PERMANENT
     6       ,NEW
     7       ,WRITE
     8       ,SIZE
     9       ,TEMPFILENUM
     A       ,PERMFILENUM
     B       ,OUTFILENUM
     C       ,STATUS
C
      character TEMPFILE*10
     2         ,PERMFILE*10
     3         ,OUTFILE*10
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
```

```
C
      DESIGNATOR  = 2
      DOMAIN      = 3
      ACCESS      = 11
      RECORD_SIZE = 19
C
      TEMPFILE = '%TEMPEMP%'
      PERMANENT = 1
      call HPFOPEN (TEMPFILENUM, STATUS, DESIGNATOR,
     2             ,TEMPFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
        print *,'HPFOPEN error on TEMPEMP.  Terminating.'
        call QUIT (1)
      endif
C
      PERMFILE = '%PERMEMP%'
      call HPFOPEN (PERMFILENUM, STATUS, DESIGNATOR,
     2             ,PERMFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
        print *,'HPFOPEN error on PERMEMP.  Terminating.'
      endif
C
      NEW   = 4
      WRITE = 1
      SIZE  = 80
      OUTFILE = '%ALLEMP%'
      call HPFOPEN (OUTFILENUM, STATUS, DESIGNATOR, OUTFILE
     2             ,DOMAIN, NEW, ACCESS, WRITE, RECORD_SIZE
     3             ,SIZE)
      if (STATUS .ne. 0) then
        print *,'HPFOPEN error on ALLEMP.  Terminating.'
      endif
C
      return
      end
C
      subroutine DO_MERGE
C
      system intrinsic HPMERGEINIT
     2                ,HPMERGEERRORMESS
     3                ,HPMERGEEND
C
      integer KEYS_ONLY
     2       ,NUMKEYS
     3       ,LENGTH
     4       ,INPUTFILES(3)
     5       ,OUTPUTFILE(2)
     6       ,KEYS(4)
     7       ,TEMPFILENUM
     8       ,PERMFILENUM
     9       ,OUTFILENUM
     A       ,STATUS
     B       ,STATISTICS(6)
C
      character ALTSEQ*2
     2         ,MESSAGE*80
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      INPUTFILES(1) = TEMPFILENUM
      INPUTFILES(2) = PERMFILENUM
      INPUTFILES(3) = 0

      OUTPUTFILE(1) = OUTFILENUM
      OUTPUTFILE(2) = 0

      KEYS_ONLY     = 0
      NUMKEYS       = 1
      KEYS(1)       = 41
      KEYS(2)       = 20
      KEYS(3)       = 0
      KEYS(4)       = 0

      ALTSEQ(1:1)   = CHAR(255)
      ALTSEQ(1:2)   = CHAR(255)

      call HPMERGEINIT (STATUS, INPUTFILES,, OUTPUTFILE,,
     2      KEYS_ONLY, NUMKEYS, KEYS, ALTSEQ)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPMERGEERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif
      call HPMERGEEND (STATUS,STATISTICS)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPMERGEERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif

      return
      end
C
      subroutine CLOSE_FILES
C
      system intrinsic FCLOSE
C
      integer*2 DISPOSITION
     2         ,SECURITYCODE
C
      integer TEMPFILENUM
     2       ,PERMFILENUM
     3       ,OUTFILENUM
     4       ,STATUS
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      DISPOSITION = 0
      SECURITYCODE = 0
C
      call FCLOSE (TEMPFILENUM, DISPOSITION, SECURITYCODE)
      call FCLOSE (PERMFILENUM, DISPOSITION, SECURITYCODE)
      DISPOSITION = 1
      call FCLOSE (OUTFILENUM, DISPOSITION, SECURITYCODE)
      return
      end
```

When this program is executed, the output is written to ALLEMP. To view ALLEMP:

```
:print allemp

Jones,              Eliza               000001              06/06/87
Gangley,            Tomas               000003              06/06/87
Smith,              James               000005              06/06/87
Jackson,            Jonathan            000006              06/06/87
Rields,             Evelyn              000007              07/12/87
Washington,         Lois                000014              07/23/87
Jackson,            Rosa                000022              08/15/87
Everett,            Joyce               000029              10/19/87
```

Example of Record Input

The following program sorts the personnel files shown below. They are sorted by last name. The program marks the employee numbers for the temporary employees with an asterisk.

The files that are used in the following example are as follows (data descriptions and character positions are indicated for convenience only):

`TEMPEMP` Information file about temporary employees:

```
Last Name           First Name       Employee Number      Hire Date

Gangley,            Tomas               000003              06/06/87
Rields,             Evelyn              000007              07/12/87
Everett,            Joyce               000029              10/19/87

0        1         2         3         4         5         6         7

1234567890123456789012345678901234567890123456789012345678901234567890
```

`PERMEMP` Information file about permanent employees:

```
Last Name          First Name        Employee Number      Hire Date

Jones,              Eliza               000001              06/06/87
Smith,              James               000005              06/06/87
Jackson,            Johnathon           000006              06/06/87
Washington,         Lois                000014              07/23/87
Jackson,            Rosa                000022              08/15/87

0        1         2         3         4         5         6         7

1234567890123456789012345678901234567890123456789012345678901234567890
```

Example D-3. SORTREC_INPUT Program

```
$standard_level system
      program SORTREC_INPUT
C
C     This program reads the files TEMPEMP and PERMEMP, alters the TEMPEMP
C  records, passes all records to SORT/XL, and outputs to the file ALLEMP.
C
      integer TEMPFILENUM
     2       ,PERMFILENUM
     3       ,OUTFILENUM
     4       ,STATUS
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      call OPEN_FILES
      call DO_SORT
      call CLOSE_FILES
      stop
      end
C
      subroutine OPEN_FILES
C
      system intrinsic HPFOPEN
     2                ,QUIT
C
      integer DESIGNATOR
     2       ,DOMAIN
     3       ,ACCESS
     4       ,RECORD_SIZE
     5       ,PERMANENT
     6       ,NEW
     7       ,WRITE
     8       ,SIZE
     9       ,TEMPFILENUM
     A       ,PERMFILENUM
     B       ,OUTFILENUM
     C       ,STATUS
C
      character TEMPFILE*10
     2         ,PERMFILE*10
     3         ,OUTFILE*10
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
C
      DESIGNATOR  = 2
      DOMAIN      = 3
      ACCESS      = 11
      RECORD_SIZE = 19
C
      TEMPFILE = '%TEMPEMP%'
      PERMANENT = 1
      call HPFOPEN (TEMPFILENUM, STATUS, DESIGNATOR,
     2             ,TEMPFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
```

```
          print *, 'HPFOPEN error on TEMPFILE.  Terminating.'
        endif
C
        PERMFILE = '%PERMEMP%'
        call HPFOPEN (PERMFILENUM, STATUS, DESIGNATOR,
     2              ,PERMFILE, DOMAIN, PERMANENT)
        if (STATUS .ne. 0) then
          print *, 'HPFOPEN error on PERMEMP.  Terminating.'
          call QUIT (2)
        endif
C
        NEW   = 4
        WRITE = 1
        SIZE  = 80

        OUTFILE = '%ALLEMP%'
        call HPFOPEN (OUTFILENUM, STATUS, DESIGNATOR,
     2              ,OUTFILE, DOMAIN, NEW, ACCESS, WRITE,
     3              ,RECORD_SIZE, SIZE)
        if (STATUS .ne. 0) then
          print *, 'HPFOPEN error on ALLEMP.  Terminating.'
          call QUIT (3)
        endif
C
        return
        end
C
        subroutine DO_SORT
C
        system intrinsic HPSORTINIT
     2                 ,HPSORTERRORMESS
     3                 ,HPSORTEND
     4                 ,HPSORTINPUT
     5                 ,FREAD
     6                 ,QUIT
C
        integer OUTPUT_OPTION
     2         ,NUMKEYS
     3         ,LENGTH
     4         ,OUTPUTFILE(2)
     5         ,KEYS(4)
     6         ,TEMPFILENUM
     7         ,PERMFILENUM
     8         ,OUTFILENUM
     9         ,STATUS
     A         ,RECLENGTH
C
        integer*2 LNGTH
C
        logical EOF
C
        character ALTSEQ*2
     2           ,MESSAGE*80
     3           ,BUFFER*80
C
        common /PARMS/ TEMPFILENUM, PERMFILENUM
     2               ,OUTFILENUM, STATUS
C
```

```
      OUTPUTFILE(1) = OUTFILENUM
      OUTPUTFILE(2) = 0

      OUTPUT_OPTION = 0

      RECLENGTH = 80

      NUMKEYS       = 1
      KEYS(1)       = 1
      KEYS(2)       = 20
      KEYS(3)       = 0
      KEYS(4)       = 0

      ALTSEQ(1:1)   = CHAR(255)
      ALTSEQ(2:2)   = CHAR(255)

      call HPSORTINIT (STATUS,, OUTPUTFILE, OUTPUT_OPTION
     2                ,RECLENGTH,, NUMKEYS, KEYS, ALTSEQ)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif
      LENGTH = 72
      EOF    = .false.

C    Read TEMPEMP file.  Start with a priming read.  If EOF is not found on
C  the priming read, call HPSORTINPUT to put the record into the sort, then
C  read and input until EOF is found.

      LNGTH = FREAD (TEMPFILENUM, BUFFER, LENGTH)
      if (ccode()) 10,30,20

   10 print *, 'FREAD error on TEMPFILE'
      call QUIT (10)

   20 EOF = .true.

   30 continue

      do while ( .not. EOF)
        BUFFER(40:40) = '*'
        call HPSORTINPUT (STATUS, BUFFER, LENGTH)
        if (STATUS .ne. 0) then
          MESSAGE = ' '
          call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
          print *, MESSAGE
        endif

C   Read the next record. CCG indicates EOF has been found.

        LNGTH = FREAD (TEMPFILENUM, BUFFER, LENGTH)
        if (ccode()) 40,60,50

   40   print *, 'FREAD error on TEMPFILE'
        call QUIT (40)

   50   EOF = .true.
   60   continue
      end do

C     Now read PERMEMP, as explained above.

      EOF   = .false.
      LNGTH = FREAD (PERMFILENUM, BUFFER, LENGTH)
      if (ccode()) 70,90,80
   70 print *, 'FREAD error on PERMEMP.'
      call QUIT (70)

   80 EOF = .true.

   90 continue

      do while (.not. EOF)
        call HPSORTINPUT (STATUS, BUFFER, LENGTH)
        if (STATUS .ne. 0) then
          MESSAGE = ' '
          call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
          print *, MESSAGE
        endif
        LNGTH = FREAD (PERMFILENUM, BUFFER, LENGTH)
        if (ccode()) 100,120,110

  100   print *, 'FREAD error on PERMEMP'
        call QUIT (100)

  110   EOF = .true.

  120   continue
      end do

      call HPSORTEND (STATUS)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif

      return
      end
C
      subroutine CLOSE_FILES
C
      system intrinsic FCLOSE
C
      integer*2 DISPOSITION
     2          ,SECURITYCODE
C
      integer TEMPFILENUM
     2        ,PERMFILENUM
     3        ,OUTFILENUM
     4        ,STATUS
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM
     2              ,OUTFILENUM, STATUS
```

```
C
      DISPOSITION = 0
      SECURITYCODE = 0
C
      call FCLOSE (TEMPFILENUM, DISPOSITION, SECURITYCODE)
      call FCLOSE (PERMFILENUM, DISPOSITION, SECURITYCODE)
      DISPOSITION = 1
      call FCLOSE (OUTFILENUM, DISPOSITION, SECURITYCODE)
      return
      end
```

When this program is executed, the output is written to ALLEMP. To view ALLEMP:

```
:print allemp

Everett,            Joyce               *000029             10/19/87
Gangley,            Tomas               *000003             06/06/87
Jackson,            Jonathan             000006             06/06/87
Jackson,            Rosa                 000022             08/15/87
Jones,              Eliza                000001             06/06/87
Rields,             Evelyn              *000007             07/12/87
Smith,              James                000005             06/06/87
Washington,         Lois                 000014             07/23/87
```

Example of Record Output

The following program sorts the personnel files shown for the last example. They are sorted by last name. The output records are altered before they are output.

Example D-4. SORTREC_OUTPUT Program

```
$standard_level system
      program SORTREC_OUTPUT
C
C     This program reads the files TEMPEMP and PERMEMP, sorts them by last
C  name, outputs them by record, alters the output recors, and prints the
C  record to $STDLIST.
C
      integer TEMPFILENUM
     2       ,PERMFILENUM
     3       ,STATUS
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM, STATUS
C
      call OPEN_FILES
      call DO_SORT
      call CLOSE_FILES
      stop
      end
C
      subroutine OPEN_FILES
C
      system intrinsic HPFOPEN
     2                ,QUIT
C
      integer DESIGNATOR
     2       ,DOMAIN
     3       ,ACCESS
     4       ,PERMANENT
     5       ,TEMPFILENUM
     6       ,PERMFILENUM
     7       ,STATUS
C
      character TEMPFILE*10
     2         ,PERMFILE*10
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM, STATUS
C
      DESIGNATOR  = 2
      DOMAIN      = 3
      ACCESS      = 11
C
      TEMPFILE = '%TEMPEMP%'
      PERMANENT = 1
      call HPFOPEN (TEMPFILENUM, STATUS, DESIGNATOR,
     2             ,TEMPFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
        print *, 'HPFOPEN error on TEMPFILE.  Terminating.'
        call QUIT (1)
      endif
C
      PERMFILE = '%PERMEMP%'
      call HPFOPEN (PERMFILENUM, STATUS, DESIGNATOR,
     2             ,PERMFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
        print *, 'HPFOPEN error on PERMEMP.  Terminating.'
        call QUIT (2)
      endif
C
      return
      end
C
      subroutine DO_SORT
C
      system intrinsic HPSORTINIT
     2                ,HPSORTERRORMESS
     3                ,HPSORTEND
     4                ,HPSORTINPUT
     5                ,HPSORTOUTPUT
     6                ,QUIT
C
      integer OUTPUT_OPTION
     2       ,NUMKEYS
     3       ,LENGTH
     4       ,INPUTFILES(3)
     5       ,KEYS(4)
     6       ,TEMPFILENUM
     7       ,PERMFILENUM
     8       ,STATUS
C
      character ALTSEQ*2
     2         ,MESSAGE*80
     3         ,BUFFER*80
C
      common /PARMS/ TEMPFILENUM, PERMFILENUM, STATUS
C
      INPUTFILES(1) = TEMPFILENUM
      INPUTFILES(2) = PERMFILENUM
      INPUTFILES(3) = 0
      LENGTH        = 1
C
      OUTPUT_OPTION = 0
C
      NUMKEYS       = 1
      KEYS(1)       = 1
      KEYS(2)       = 20
      KEYS(3)       = 0
      KEYS(4)       = 0
C
      ALTSEQ(1:1)   = CHAR(255)
      ALTSEQ(2:2)   = CHAR(255)
C
      call HPSORTINIT (STATUS, INPUTFILES,, OUTPUT_OPTION
     2                ,,, NUMKEYS, KEYS, ALTSEQ)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
```

```
             print *,MESSAGE
           endif
     C
           do while (LENGTH .gt. 0)
             call HPSORTOUTPUT (STATUS, BUFFER, LENGTH)
           BUFFER(33:39) = 'Empl. #'
           BUFFER(50:59) = 'Hire Date:'
             print *,BUFFER
             if (STATUS .ne. 0) then
               call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
               print *,MESSAGE
             endif
           end do
     C
           call HPSORTEND (STATUS)
           if (STATUS .ne. 0) then
             MESSAGE = ' '
             call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
             print *,MESSAGE
           endif
     C
           return
           end
     C
           subroutine CLOSE_FILES
     C
           system intrinsic FCLOSE
     C
           integer*2 DISPOSITION
          2         ,SECURITYCODE
     C
           integer TEMPFILENUM
          2       ,PERMFILENUM
          3       ,STATUS
     C
           common /PARMS/ TEMPFILENUM, PERMFILENUM, STATUS
     C
           DISPOSITION  = 0
           SECURITYCODE = 0
     C
           call FCLOSE (TEMPFILENUM, DISPOSITION, SECURITYCODE)
           call FCLOSE (PERMFILENUM, DISPOSITION, SECURITYCODE)
     C
           return
           end
```

When this program is executed, the output is written to the screen:

```
Everett,            Joyce       Empl. # 000029   Hire Date: 10/19/87
Gangley,            Tomas       Empl. # 000003   Hire Date: 06/06/87
Jackson,            Jonathan    Empl. # 000006   Hire Date: 06/06/87
Jackson,            Rosa        Empl. # 000022   Hire Date: 08/15/87
Jones,              Eliza       Empl. # 000001   Hire Date: 06/06/87
Rields,             Evelyn      Empl. # 000007   Hire Date: 07/12/87
Smith,              James       Empl. # 000005   Hire Date: 06/06/87
Washington,         Lois        Empl. # 000014   Hire Date: 07/23/87
```

Example of Using an Altered Sequence

The following example sorts the data file below, DATA. The entries in DATA are sorted using an altered collating sequence that is explicitly specified in the program. The sequence contains all displayable ASCII characters and alters the order of the alphabetic characters to AaBbCc .... The output file is called FRUIT

DATA	File of fruit names

```
banana
Apple
Grapes
grapes
Pear
peach
orange
```

Example D-5. SORTALT Program

```
$standard_level system
      program SORTALT
C
C     This program reads the files TEMPEMP and PERMEMP, sorts them by last
C  name, outputs them by record, alters the output recors, and prints the
C  record to $STDLIST.
C
      integer DATAFILENUM
     2       ,FRUITFILENUM
     3       ,STATUS
C
      common /PARMS/ DATAFILENUM, FRUITFILENUM, STATUS
C
      call OPEN_FILES
      call DO_SORT
      call CLOSE_FILES
      stop
      end
C
      subroutine OPEN_FILES
C
      system intrinsic HPFOPEN
     2                ,QUIT
C
      integer DESIGNATOR
     2       ,DOMAIN
     3       ,ACCESS
     4       ,PERMANENT
     5       ,DATAFILENUM
     6       ,FRUITFILENUM
     7       ,STATUS
     8       ,RECORD_SIZE
     9       ,NEW
     A       ,WRITE
     B       ,SIZE
C
      character DATAFILE*10
     2         ,FRUITFILE*10
C
      common /PARMS/ DATAFILENUM, FRUITFILENUM, STATUS
C
      DESIGNATOR  = 2
      DOMAIN      = 3
      ACCESS      = 11
      RECORD_SIZE = 19
C
      DATAFILE = '%DATA%'
      PERMANENT = 1
      call HPFOPEN (DATAFILENUM, STATUS, DESIGNATOR,
     2             ,DATAFILE, DOMAIN, PERMANENT)
      if (STATUS .ne. 0) then
        print *, 'HPFOPEN error on DATAFILE.  Terminating.'
        call QUIT (1)
      endif
C
```

```
      NEW       = 4
      WRITE     = 1
      SIZE      = 80
      FRUITFILE = '%FRUIT%'
      call HPFOPEN (FRUITFILENUM, STATUS, DESIGNATOR,
     2             ,FRUITFILE, DOMAIN, NEW, ACCESS, WRITE
     3             ,RECORD_SIZE, SIZE)
      if (STATUS .ne. 0) then
        print *, 'HPFOPEN error on FRUITFILE. Terminating.'
        call QUIT (2)
      endif
C
      return
      end
C
      subroutine DO_SORT
C
      system intrinsic HPSORTINIT
     2                ,HPSORTERRORMESS
     3                ,HPSORTEND
     4                ,HPSORTINPUT
     5                ,HPSORTOUTPUT
     6                ,QUIT
C
      integer OUTPUT_OPTION
     2       ,NUMKEYS
     3       ,INPUTFILES(2)
     4       ,OUTPUTFILE(2)
     5       ,KEYS(4)
     6       ,DATAFILENUM
     7       ,FRUITFILENUM
     8       ,STATUS
C
      character ALTSEQ*96
     1         ,MESSAGE*80
C
      common /PARMS/ DATAFILENUM, FRUITFILENUM, STATUS
C
      INPUTFILES(1) = DATAFILENUM
      INPUTFILES(2) = 0
C
      OUTPUTFILE(1) = FRUITFILENUM
      OUTPUTFILE(2) = 0
C
      OUTPUT_OPTION = 0
C
      NUMKEYS       = 1
      KEYS(1)       = 1
      KEYS(2)       = 20
      KEYS(3)       = 0
      KEYS(4)       = 0
C
      ALTSEQ(1:2)   = '  '
      ALTSEQ(1:1)   = CHAR(0)
      ALTSEQ(2:2)   = CHAR(93)
C
      ALTSEQ(3:17)  = '!"#$%&''()*+,-./'
      ALTSEQ(18:33) = '0123456789::<=>?'
      ALTSEQ(34:49) = '@AaBbCcDdEeFfGgH'
      ALTSEQ(50:65) = 'hIiJjKkLlMmNnOoP'
      ALTSEQ(66:80) = 'pQqRrSsTtUuVvWwX'
      ALTSEQ(81:95) = 'xYyZz[\]^^_{|}~'
C
      call HPSORTINIT (STATUS, INPUTFILES, OUTPUTFILE
     2                ,OUTPUT_OPTION, ,,, NUMKEYS, KEYS
     3                ,ALTSEQ,,,STATISTICS)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif
C
      call HPSORTEND (STATUS)
      if (STATUS .ne. 0) then
        MESSAGE = ' '
        call HPSORTERRORMESS (STATUS, MESSAGE, LENGTH)
        print *,MESSAGE
      endif
C
      return
      end
C
      subroutine CLOSE_FILES
C
      system intrinsic FCLOSE
C
      integer*2 DISPOSITION
     2         ,SECURITYCODE
C
      integer DATAFILENUM
     2       ,FRUITFILENUM
     3       ,STATUS
C
      common /PARMS/ DATAFILENUM, FRUITFILENUM, STATUS
C
      DISPOSITION  = 0
      SECURITYCODE = 0
C
      call FCLOSE (DATAFILENUM, DISPOSITION, SECURITYCODE)
      call FCLOSE (FRUITFILENUM, DISPOSITION, SECURITYCODE)
C
      return
      end
```

When this program is executed, the output is written to FRUIT. To view FRUIT:

```
:print fruit

Apple
banana
Grapes
grapes
peach
Pear
orange
```

E Data Types

When you sort or merge data with SORT-MERGE/XL, your sorting key is one of many data types. This appendix explained the format of SORT-MERGE/XL's generic data types and gives language equivalents of them.

For more information about data types, refer to *Data Types Conversion Programmer's Guide*.

In `HPSORTINIT` or `HPMERGEINIT` you specify the type of data that is sorted by your keys. The data type is given in the third element of the *keys* parameter array. For more information about specifying keys, refer to Chapter 2.

This appendix tells you the language equivalents of SORT-MERGE/XL generic data types. It covers the following languages:

- HP Business BASIC
- HP C/XL
- HP COBOL II/XL
- HP FORTRAN 77/XL
- HP Pascal

The following tables show the *keys* parameter value for each SORT-MERGE/XL data type. For example, 0 is specified in the *keys* parameter array if the key is ASCII or EBCDIC byte type.

HP Business BASIC

Table E-1. Business BASIC Equivalents of SORT-MERGE/XL Data Types

	SORT-MERGE/XL Data Type	Business BASIC Equivalent
0	Byte (character)	STRING$[*n*]
1	Twos complement (integer)	
	2-byte (short integer)	SHORT INTEGER
	4-byte (integer)	INTEGER
2	HP 3000 floating point	
	4-byte (real)	SHORT REAL
	8-byte (double precision)	REAL
10	Character	STRING$[*n*]
12	Short floating point Decimal	SHORT DECIMAL
13	floating point Decimal	DECIMAL

NOTE The variable *n* refers to an integer number.

HP C/XL

Table E-2. HP C XL Equivalents of SORT-MERGE/XL Data Types

	SORT-MERGE/XL Data Type	C Equivalent
0	Byte (character)	CHAR
1	Twos Complement (integer)	
	2-byte (short integer)	SHORT
	4-byte (integer)	INT, LONG
2	HP 3000 floating point	
	4-byte (real)	FLOAT
	8-byte (double precision)	DOUBLE
3	IEEE Standard floating point	
	4-byte (real)	FLOAT
	8-byte (double precision)	DOUBLE
10	Character	CHAR

Floating point data may be stored in HP 3000 or IEEE Standard format:

- IEEE Standard floating point data is the default MPE XL NM (Native Mode) floating point format.
- HP 3000 floating point data may also be created by explicit specification in NM. HP 3000 floating point data is the default in MPE XL CM (Compatibility Mode) or on an MPE V-based system.

HP COBOL II/XL

Table E-3. HP COBOL II/XL Equivalents of SORT-MERGE/XL Data Types

	SORT-MERGE/XL Data Type	HP COBOL II/XL Equivalent
0	byte (character)	PIC X(*n*) or a group
1	twos complement (integer)	
	2-byte (short integer)	S9(*n*) COMP (where $1 \leq \nu \leq 4$)
	4-byte (integer)	S9(*n*) COMP (where $5 \leq \nu \leq 9$)
	8-byte (long integer)	S9(*n*) COMP (where $10 \leq \nu \leq 18$)
4	packed decimal with odd number of digits	S9(*n*) COMP-3 (where *n* is an odd number)
5	packed decimal with even number of digits	S9(*n*) COMP-3 (where *n* is an even number)
6	display trailing sign	S9(*n*) SIGN IS TRAILING
7	display leading sign	S9(*n*) SIGN IS LEADING
8	display trailing sign separate	S9(*n*) SIGN IS TRAILING SEPARATE
9	Display-Leading-Sign-Separate	S9(*n*) SIGN IS LEADING SEPARATE
10	Character	PIC X(*n*) or a group

NOTE The variable *n* refers to an integer number.

HP FORTRAN 77/XL

Table E-4. HP FORTRAN 77/XL Equivalents of SORT-MERGE/XL Data Types

	SORT-MERGE/XL Data Type	HP FORTRAN 77/XL Equivalent
0	byte (character)	CHARACTER**n*
1	twos complement (integer)	
	2-byte (short integer)	INTEGER*2
	4-byte (integer)	INTEGER*4
2	HP 3000 floating point	
	4-byte (real)	REAL*4
	8-byte (double precision)	REAL*8
3	IEEE Standard floating point:	
	4-byte (real)	REAL*4
	8-byte (double precision)	REAL*8
10	Character	CHARACTER**n*

NOTE The variable *n* refers to an integer number.

Floating point data may be stored in HP 3000 or IEEE Standard format:

- IEEE Standard floating point data is the default MPE XL NM (Native Mode) floating point format.
- HP 3000 floating point data may also be created by explicit specification in NM. HP 3000 floating point data is the default in MPE XL CM (Compatibility Mode) or on an MPE V-based system.

HP Pascal

Table E-5. HP Pascal Equivalents of SORT-MERGE/XL Data Types

	SORT-MERGE/XL Data Type	HP Pascal Equivalent
0	byte (character)	CHAR
1	twos complement (integer):	
	2-byte (short integer)	SHORTINT
	4-byte (integer)	INTEGER
2	HP 3000 floating point:	
	4-byte (real)	REAL
	8-byte (double precision)	LONGREAL
3	IEEE standard floating point:	
	4-byte (real)	REAL
	8-byte (double precision)	LONGREAL
10	character	CHAR

Floating point data may be stored in HP 3000 or IEEE Standard format:

- IEEE Standard is the default format for floating point data in MPE XL NM (Native Mode).
- HP 3000 is the default format for floating point data in MPE XL CM (Compatibility Mode) or on an MPE V-based system. HP 3000 format may also be created in NM by explicit specification.

For further information, refer to *Data Types Conversion Programmer's Guide* (32650-90015)

Index

Index