Nsort™
User Guide

Release 3.5

4 December 2010

Nsort User Guide
Release 3.5.1

Contents

Preface

This preface provides an overview to the *Nsort User Guide,* including sections on:

- The intended audience for this document.
- How this document is organized.
- The notational conventions used in this document.
- How to find more information about Nsort.

Intended Audience

Nsort User Guide is written for database administrators programmers who use Nsort as part of their data warehousing management systems and system administrators who manage large sort applications.

The document assumes that you are familiar with:

- Relational database.
- Your RDBMS vendor's products.

How this Document is Organized

Chapter 1, "Introduction"
This chapter provides an introduction to the features and functionality provided by the Nsort product.

Chapter 2, "The Nsort Command Line"
This chapter describes the Nsort standard and POSIX sort-like command lines.

Chapter 3, "Specification Language Overview"
This chapter discusses Nsort sort specification statements, and explains how to use sort specification files with Nsort.

Chapter 4, "Describing the Sort Data"
This chapter explains how to describe records and fields to Nsort.

Chapter 5, "Sort Definition Statements"
This chapter discusses how to sort, select, reformat and merge records.

Chapter 6, "Configuration and Performance"
This chapter discusses using Nsort's configuration and performance options.

Chapter 7, "Sort Subroutine Library"
This chapter presents the Nsort subroutine library which provides a sort API for application programs.

Notational Conventions

The following conventions are used throughout this document:

Convention	Description
cd ..	A **command** to be typed exactly as it appears in the document.
nsort *filename*	A *placeholder* for a user-supplied value.
/data/spec	The name of a *`file`* or a *`directory path`*.
`-summarize=total`	`Example` command lines or specification statements.
NSORT	The name of an UNIX ENVIRONMENT VARIABLE.
TRUE	The RESULT VALUE of a conditional expression.

For More Information

For additional information on Nsort or technical support for Nsort, you can contact Ordinal Technology as follows:

- By telephone: 925-253-9204.
- By email: info@ordinal.com or support@ordinal.com.
- World-wide web: http://www.ordinal.com.

Introduction 1

Nsort™ is a multiprocessor, high speed sort utility and subroutine library for UNIX™ and Windows™ systems. Nsort processes files at high speeds by simultaneously using many processors and disks. Nsort uses efficient, proprietary algorithms to sort records and reduce disk I/O waits commonly associated with processing very large data sets. Nsort speeds up both the processing of data outside a database system and the loading of data into a database or data warehouse.

The rapid growth of the Internet, demand for more historical business data, and falling disk prices are resulting in ever-larger corporate data sets. Analyzing this data is an increasing challenge. Today various Internet, credit bureau, direct marketing, telephone and utility companies, as well as government agencies, are using Nsort to quickly sort, merge, and summarize their large data sets.

Nsort provides the sort features needed to presort load data for data warehouse installations. Nsort can dramatically improve the performance of your data warehouse loads.

This chapter introduces Nsort and provides a brief overview to the Nsort product. This chapter discusses the following topics:

- How to put Nsort to work for you.
- What Nsort can do.

Putting Nsort to Work for You

If you are a database administrator responsible for large amounts of data, sort performance affects many parts of your system. Nsort can improve the speed of your bulk data transfer operations in the following ways:

- Presort the incoming data by primary key so that the RDBMS can append the data sequentially. If the table has no pre-existing primary index, presorting expedites the creation of the primary index. Even if the table already has a primary index, presorting often speeds the bulk data load.
- Merge sorted data from multiple sources into one sorted file that can be quickly copied into your data warehouse.
- Filter out unneeded or improperly-formed records before the RDBMS loader even sees them.
- Summarize input data so that the RDBMS can directly load the totaled data into a summary table.
- Prepare load data in advance of the load window, thus effectively shortening the time you need to keep your data warehouse off-line.
- Make the most of your current hardware. With Nsort's help, you can get your data loaded more quickly without having to purchase more processors or memory.

What Nsort Can Do

Nsort sorts, merges, summarizes, or concatenates files. Nsort can perform its functions much faster than the standard UNIX **sort** or other sort products.

Nsort allows you to customize the sort so that it suits the needs of your application. Nsort can perform the following functions:

- Sort large data sets quickly.
- Merge pre-sorted input files into one sorted output file.
- Subtotal fields by key value and store the subtotal records in the output file.
- Select records to include or omit using conditional criteria.
- Use selection criteria to create multiple sort output files.
- Sort by binary data keys as well as character data keys.
- Sort files with records and keys up to 65,535 bytes wide (or 8 megabytes for the Windows versions of Nsort).
- Handle input and output files as large as the underlying file system supports, without any 4GB limit.
- Sort on an unlimited number of sort keys.
- Read sort input from multiple input files.
- Accept command lines for the POSIX sort utility included with most UNIX implementations.
- Supports fixed-length, variable (length-prefix) and delimited (lines of text) record types.
- Use multiple processors and disks in parallel.
- Add, drop, and reorder the fields in the sort records.
- Delete duplicate records.
- Perform an ascending or descending sort on each key.
- Use large main memories with 64-bit addressing to sort data without using temporary files.

This section introduces the major functions of Nsort and includes the subsections discussing the following functions:

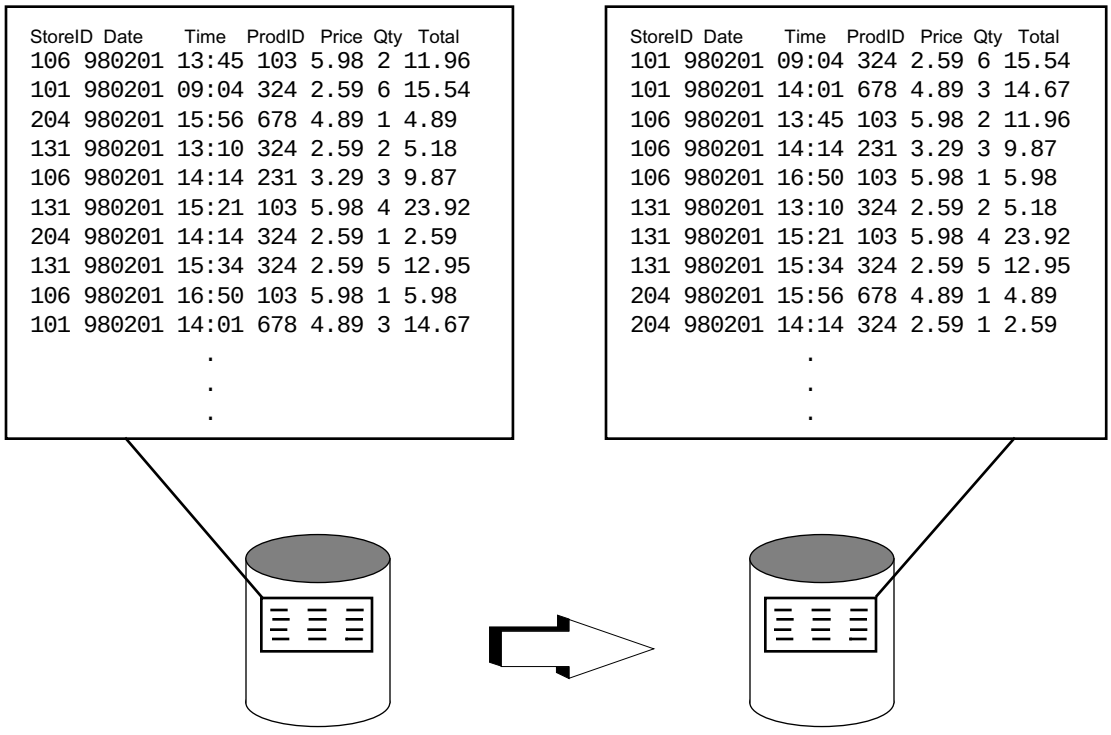
- Sorting data.
- Merging pre-sorted input streams into one output stream.
- Summarizing data.
- Modifying the action of your sort, merge, or summarize.

Sorting Data

The sort operation takes one or more input files containing records and orders the records by key value into one or more output files. Nsort supports 7 different data key types: integer, unsigned integer, character, IEEE floating point, double precision floating point, decimal character, and month.

ABC Stores adds the daily receipt data from its stores to its centralized data warehouse. Before loading the data into the data warehouse, the database administrator uses Nsort to presort the data by the data warehouse key values.

Figure 1-1 Basic Sort (by StoreID)



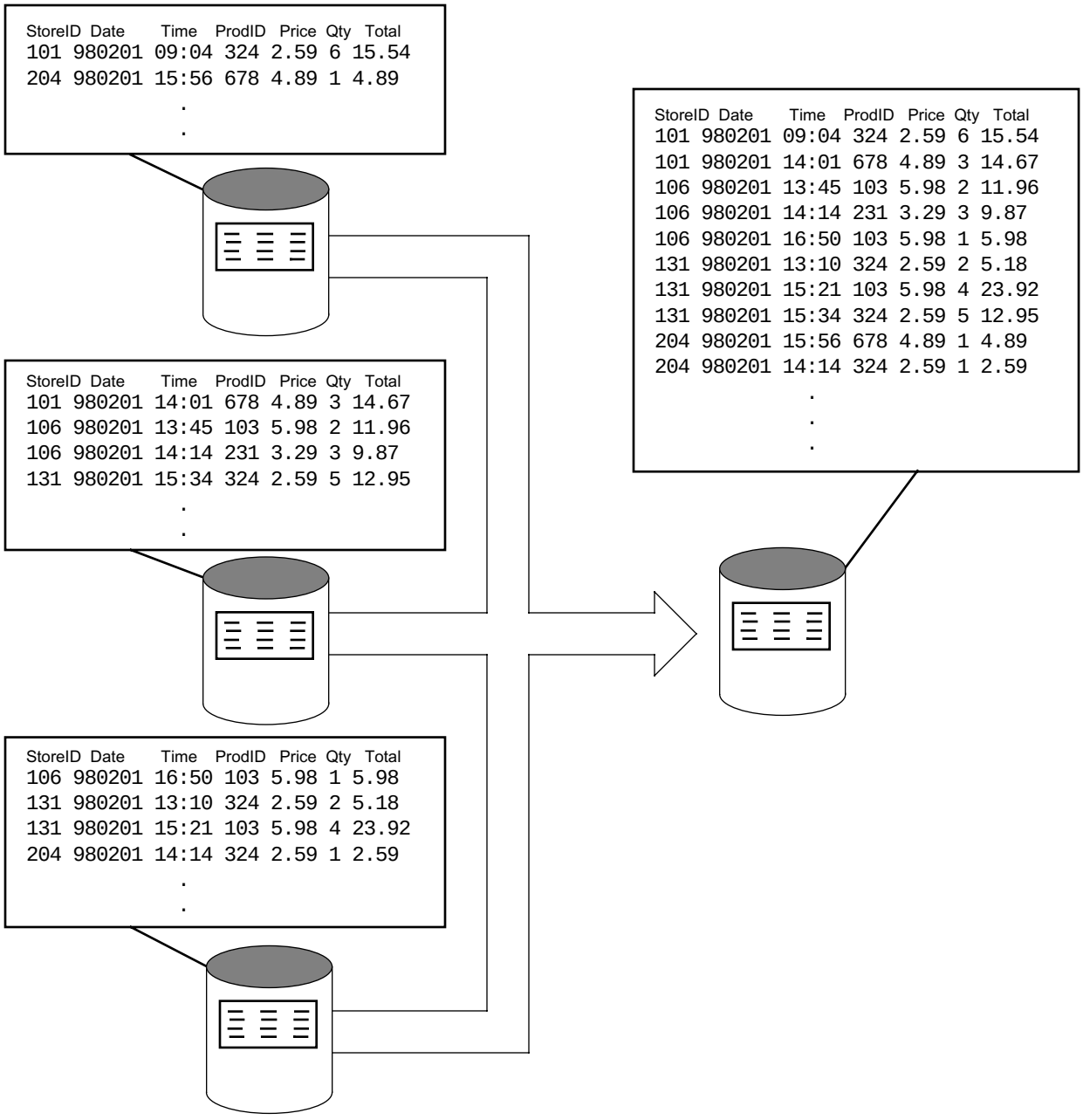
Merging Input Streams

The merge operation takes several sorted input files and merges them to produce one or more sorted output files.

XYZ Stores adds the daily receipt data from its stores to its centralized data warehouse. The receipt data is sorted by data warehouse key values at the individual stores and sent on tape to the centralized data warehouse. The database administrator uses Nsort's merge feature to merge the presorted data files into one sorted file for the bulk data load.

The next page contains a diagram of a merge operation.

Figure 1-2 Merging Input Streams

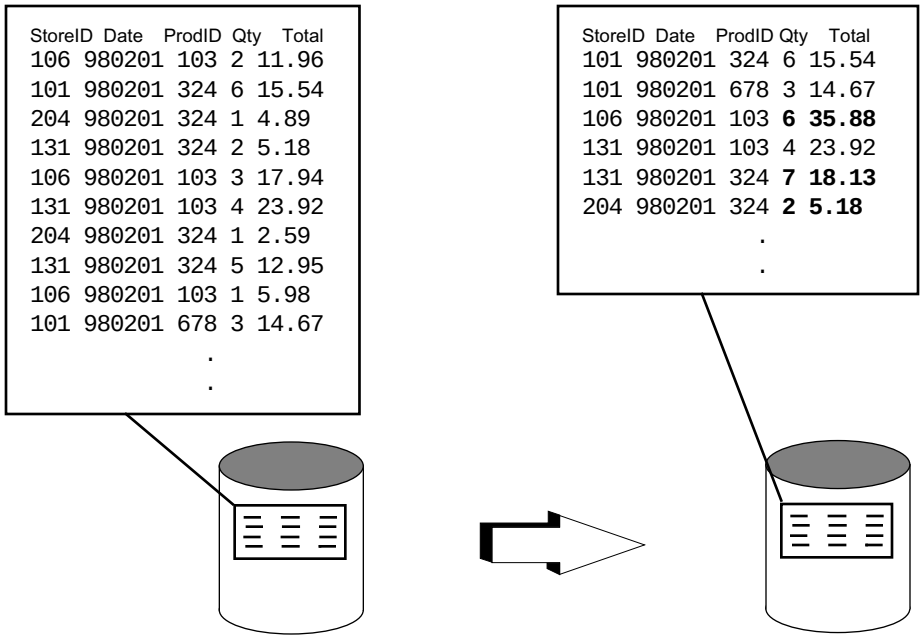


Summarizing Fields

Nsort can create summary files containing records subtotaled by key value. Use summarize on either a sort or merge operation to subtotal data fields for summary tables in your data warehouse.

ABC Stores maintains summary tables in their data warehouse to track the sales of products by store and date. The database administrator uses Nsort's summarize feature to create the summary table data from the daily receipt records.

Figure 1-3 Summarize Data



Modifying Sort Actions

Nsort provides many options to customize your sort, merge, or summarize operation. Nsort's features include the following:

- Record definitions that describe the form of your input records.
- Field definitions that describe the form of your input data fields.
- Key definitions to specify the key fields and the order and direction in which to sort them.
- Record layout specification that describe exactly how you want your output records to look.
- Derived fields from the values of other fields in your data set.
- Record selection, allowing you to include or omit records from your sort.
- Duplicate key processing.

This section discusses the options you can use to modify your sort, merge, or summarize:

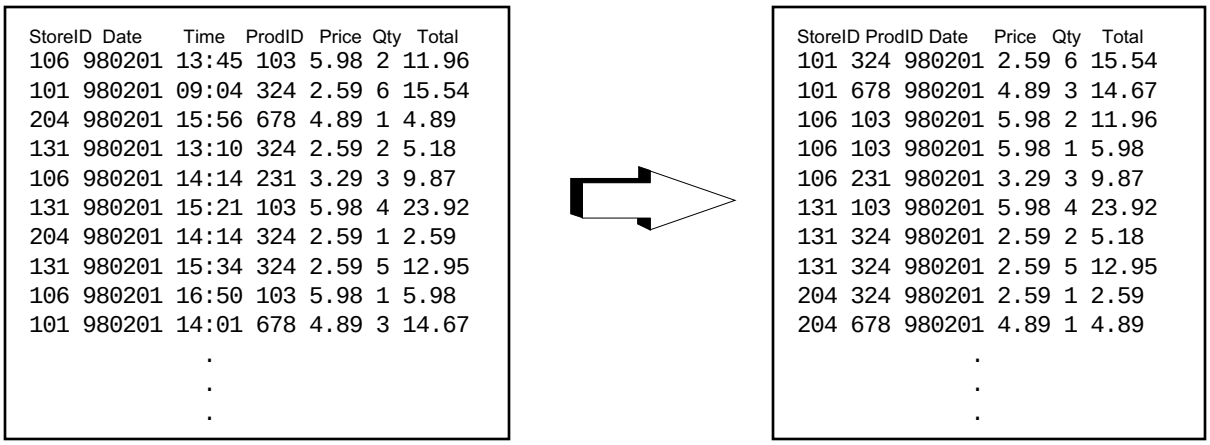
- Specifying which fields in the record you want to include in your sort or merge and in each output file.
- Deriving new fields from the values of other fields.
- Creating multiple output files with different characteristics.
- Selecting records to include or omit from each output files.
- Deleting duplicate records from the output data.
- Tuning the performance of individual sorts.

Reformatting Records

Nsort's reformat option provides control over your record layout. You can reformat the input data or the records to be placed in any output file. You can specify the fields to include in the record, the order of the fields in the record, and whether to include any derived fields or not.

The ABC Stores data warehouse receives data fields from its store that it does not track in its data warehouse applications. When the database administrator presorts the data prior to bulk loading, the database administrator reformats the records to include just those fields that are used in the data warehouse applications.

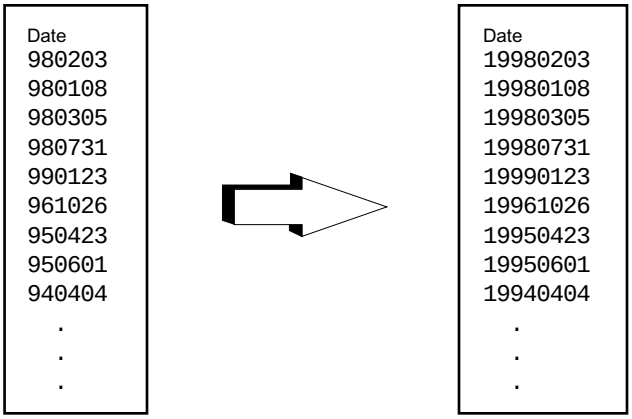
Figure 1-4 Reformat Records



Deriving New Fields

Nsort allows you to derive new fields from those existing in the input file. You can use this option to normalize the values in your database or to change the form of the data. For example, you might need to change the year field in your database from `98` to `1998`.

Figure 1-5 Deriving New Fields

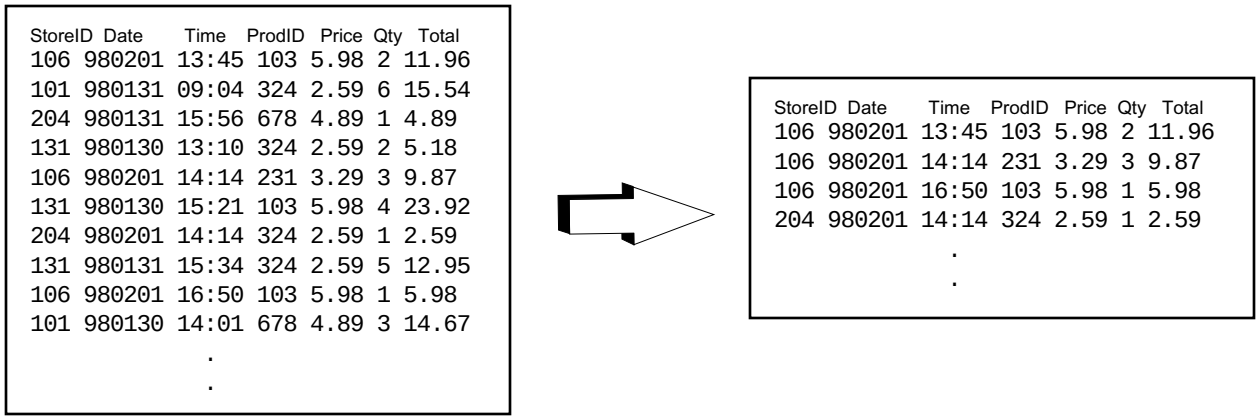


Selecting Records

Nsort allows you to select records for inclusion or omission from each output file.

XYZ Stores' individual stores sometimes mistakenly send data for the previous day along with the current day's receipts. The database administrator uses Nsort's record selection feature to include only the receipts for the current day in the bulk load input file.

Figure 1-6 Selecting Records

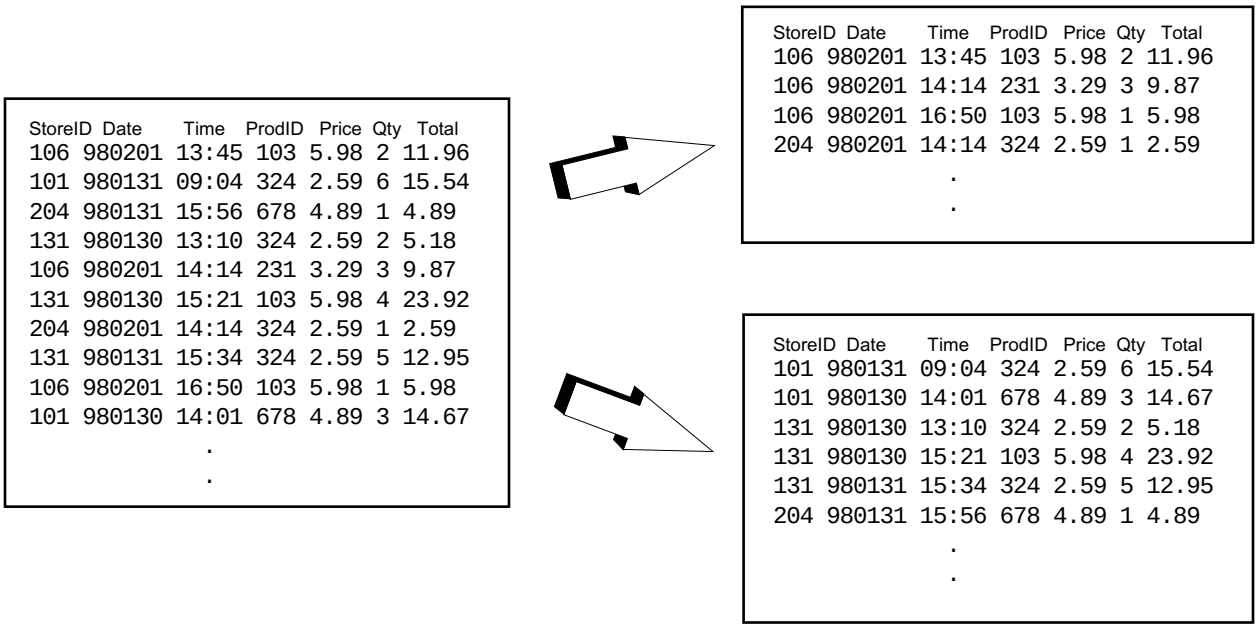


Multiple Output Files

Nsort supports the creation of multiple output files from a single sort. This allows you to apply different criteria to each output file to meet the needs of different parts of your application for the same data.

ABC Stores has a problem with individual stores sending incomplete or incorrect records as part of the data to be loaded into the data warehouse. When Nsort presorts the load data, the database administrator uses multiple files and record selection to filter good records into the bulk load input file and bad records into a rejects file.

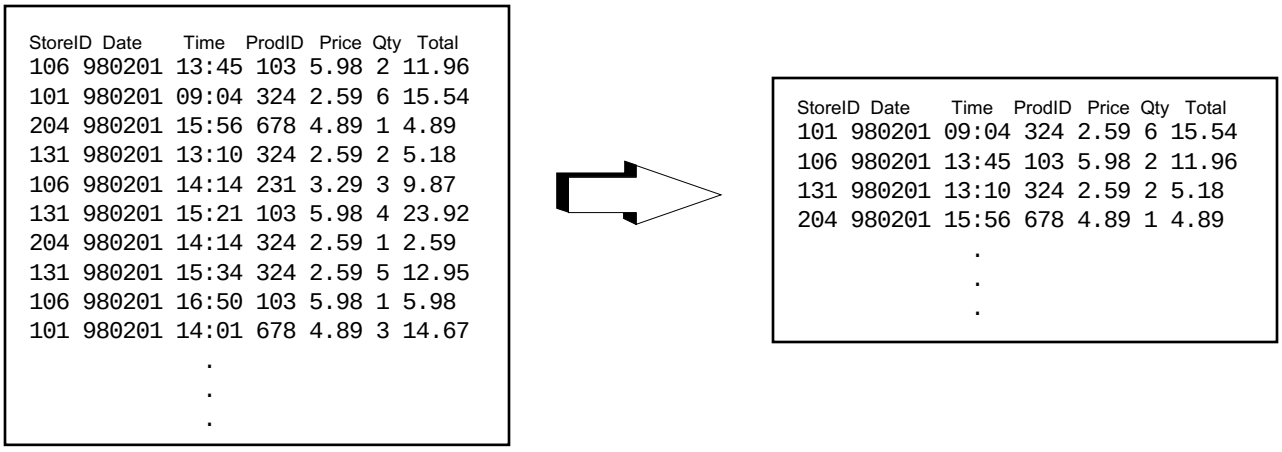
Figure 1-7 Multiple Output Files1



Deleting Duplicates

By default, Nsort retains all copies of equally-keyed records. If you want Nsort to enforce uniqueness of key values, you can tell Nsort to delete records with key values that match previously-processed records.

Figure 1-8 Deleting Duplicates



Performance Tuning

Nsort was designed with speed in mind. With normal data distribution (like the data distribution you find in a typical application), Nsort chooses the most efficient performance options on its own. For the vast majority of sort applications, you'll never need to set Nsort's performance options. Nsort does, however, provide performance tuning options for those few cases when tweaking the sort strategy can improve performance.

Nsort provides the following performance tuning and evaluation options:

- Memory usage.
- Number of processes.
- Sort methods.
- Disk I/O.
- Sort statistics.

For more information on configuration and performance options, refer to Chapter 6, "Configuration and Performance".

The Nsort Command Line 2

To invoke Nsort, you can either use the standard Nsort command line or Nsort's POSIX-sort compatible command line. Refer to "POSIX Sort Compatible Command Line" on page 23 for information on using the POSIX-sort compatible command line. The Windows version of Nsort also accepts most command line arguments of the standard Windows sort program - see "Windows Sort Compatible Command Line" on page 26. When using the standard Nsort command line, you describe your sort with Nsort's specification language. The sort specification language tells Nsort how to process your sort. Nsort accepts sort specifications from a variety of sources, including the following:

- The system-wide Nsort default option file.
- A user-specific Nsort default option file.
- The Nsort environment variable.
- The command line.
- One or more sort specification files.

Wherever you choose to describe your sort, you use the sort specification language outlined in Chapter 3, "Specification Language Overview." Sort specification commands can be issued in the default files, in an environment variable, on the command line, or in the sort specification files. All of these statement sources use the same Nsort syntax.

Specifically, Nsort processes sort specifications in the following order:

1. The system-wide Nsort default options.
   Nsort reads the system-wide Nsort default options and applies them to the current sort. See "System-wide Default File" on page 21.

2. The user-specific Nsort default options.
   Nsort reads the user-specific default options and applies them to the current sort. Any user-specific default options that conflict with the system-wide default options take precedence over the system-wide defaults. See "User Home Directory File" on page 22.

3. Your Nsort environment variable.
   Nsort reads the commands from the NSORT environment variable and applies them to the current sort. Any environment variable commands that conflict with the default options take precedence over the default options. See "Environment Variable" on page 22.

4. The Nsort command line.
   Nsort reads the commands from the Nsort command line and applies them to the current sort. If the command line refers to any specification files, the specification file commands are processed at this time. If a command conflicts with any previously-read commands, the last command read take precedence.

This chapter describes the Nsort command line and the various ways of specifying your sort to Nsort. This chapter includes the following sections:

- The Nsort command line.
- Specification files.
- Global options.

The Nsort Command Line

Nsort supports two flavors of command line. The standard Nsort command line provides access to all of Nsort's functionality. The POSIX sort compatible command line provides a command line similar to that found in the standard POSIX sort.

This section discusses the Nsort command line and includes the following subsections:

- The standard Nsort command line.
- POSIX sort compatible command line.

Standard Command Line

The standard Nsort command line can be used to describe the entire sort or name sort specification files to describe the sort.

nsort [*data definition statements*...] [*sort definition statements*...] [*input files*...] [**-o** *output-file*]

If no parameters are given, Nsort sorts the standard input as lines of text and stores the result into the standard output.

For a list of commands accepted by Nsort on the command line, refer to Chapter 3, "Specification Language Overview."

Each Nsort specification language statement must appear as one command line argument to Nsort. Statements that contain spaces, quotes, or other characters that are interpreted by the shell need to be appropriately quoted. For example, the following statement in a specification file:

```
-derived=name= status, value="valid"
```

should appear on the Nsort command line as:

```
'-derived=name= status, value="valid"'
```

Specification Files

Sort specification files allow you to specify sorts with complex record formats and conditional processing. With sort specification files, you can use the same record, field, or key specifications for multiple sorts.

Nsort provides the capability to name sort specification files on the command line or in other sort specification files. Each sort can refer to multiple specification files.

In any of these sources, you can name text files containing some or all of the sort definition. This is especially useful for complex record formats, conditions, and other commonly reused statements.

The following example illustrates the inclusion of two specification files on the Nsort command line:

```
-specification:record_fields.spec -spec:my_temp_files.spec
```

Specification files can refer to other specification files, up to a nesting level of 20. A referenced specification file is always processed in its entirety before returning to the referencing specification file or command line for further statement processing.

The following example shows a specification file describing data fields that are used in multiple sort operations:

```
-field=amount_due, decimal, size:12,
       past_due, decimal, size:5
```

For a list of commands accepted by Nsort in specification files, refer to Chapter 3, “Specification Language Overview”.

Global Options

The Nsort global options apply to every sort performed by an individual user or on a specific system. The global options give you a way to set your default options once rather than reiterating them for each different sort.

This section discusses the global options and includes subsections on the following global options:

- The Nsort system-wide default options.
- The Nsort user-specific default options.
- The Nsort environment variable.

System-wide Default File

Nsort first checks the system-wide default options file when parsing the sort commands. If the system-wide default file is present, Nsort processes the commands in the system-wide default file. This file is found in the following locations:

- On HP-UX and Linux systems, */opt/nsort/nsort.params*
- On Linux systems, */usr/lib/nsort/nsort.params*
- On Solaris systems, */opt/ORDnsort/nsort.params*
- On Windows, the Nsort installation directory.

You can use any of the standard Nsort commands in the system-wide default options. The system-wide default options provide a convenient way to specify commands that apply to most or all of the sorts performed on your system.

For example, the following statement in the system-wide options file turns off the automatic addition of derived fields:

```
-no_add_derived
```

User Home Directory File

Nsort checks the user-specific default options when parsing the sort commands. If the file *$HOME/.nsortrc* is present, Nsort processes the commands in *$HOME/.nsortrc.*

You can use any of the standard Nsort commands in the user-specific default options. The user-specific default options provide a convenient way to specify commands that apply to most or all of your sorts.

Environment Variable

Nsort checks the NSORT environment variable when parsing the sort commands. If the environment variable is present, Nsort processes the commands in the NSORT environment variable.

You can use any of the standard Nsort commands in the NSORT environment variable.

POSIX Sort Compatible Command Line

Nsort accepts sort specifications in the style of the POSIX sort standard, the standard sort command on Unix and Linux systems, as follows:

nsort [**-b**] [**-d**] [**-f**] [**-i**] [**-*k*** [*m.n*[,*m.n*]]] [**-m**] [**-M**] [**-n**] [**-r**] [**-t**c] [**-T***tempdir*] [**-u**] [**+***m***.***n* [**-***m***.***n*]] *input_files* [**-o** *output_file*]

These arguments can be used to sort lines of text with delimited fields. POSIX sort requires that records be lines of text (delimited by the newline character). By default, the fields in each record are separated by space and tab characters.

POSIX sort arguments can only appear on Nsort's command line, not in an Nsort specification file. All Nsort definition statements except those that define records, fields or keys can be used in addition to POSIX sort arguments.

Flags:

-b Ignore leading blanks when finding keys.

-d Sort in “dictionary'' order. Just letters, digits, spaces and tabs are used for comparisons.

-f Sort lowercase letters as if they were uppercase.

-i Ignore nonprintable and multibyte characters in comparisons.

-k <*w*>[.<*x*>][**bdfiMnr**][<*y*>[.<*z*>][**b**]] Define a key from *w.x* to *y.z*.

-m Merge already-sorted input files.

-M Compare first three non-blank characters as month names.

-n Sort initial numeric strings according to their arithmetic value.

-o <*name*> Set the output file name.

-r Sort in reverse (descending) order of keys.

-t<*x*> Use <*x*> as the field separator when finding key. The default field separator is any sequence of space and tab characters.

-**T**<*dir*> Create a temporary file in the *dir* directory.

-u Unique: delete all but one in each set of lines having equal keys.

-k <*w*>[.<*x*>][**b**][**d**][**f**][**i**][**M**][**n**][**r**][<*y*>[.<*z*>][**b**]]
Define a key starting in the *w*th field (where 1 is the 1st field). If *x* is specified, the key starts at the *x*th character in the field. The ***d*, f**, **i**, **M**, **n** and **r** modifiers can be used to specify the key type for this particular key only (see above flags using the same characters). If no *y* ending field is specified, the key field ends at end of the record. Otherwise the end of the key is the end of the *y*th field, or the *z*th character in the *y*th field.
A **b** modifier for the key beginning indicates that initial blanks should be skipped when determining the beginning of the key field. A **b** modifier for the key ending indicates that blanks should be skipped when determining the *z*th character that ends the field.

-**z**<*recsz*> Set longest line length. By default, this limit is 4096 characters.

+<*m*>[.<*n*>][**b**][**d**][**f**][**i**][**M**][**n**][**r**]
Start a key <*n*> characters into field <*m*+1>. The ***d*, f**, **i**, **M**, **n** and **r** modifiers can be used to specify the key type for this particular key only (see above flags using the same characters). A **b** modifier indicates that initial blanks should be skipped when determining the beginning of the key field.
Note that lowest values of *m* and *n* are 0, whereas the *w* and *x* used with **-b** start with 1.

-<*m*>[.<*n*>][**b**] End a key at the end of the <*m*+1>th field, or the <*n*+1>th character of the <*m*+1>th field. A **b** modifier indicates that initial blanks should be ignored in determing the <*n*+1>th character.

<*files*> List of input file names.

Nsort and POSIX Sort Differences

Nsort differs from POSIX sort in the following ways:

- Nsort runs much faster for all but the smallest sorts.
- Nsort does not support the **-c** option of POSIX sort.
- Nsort does not yet support supplementary code set characters. The LANG environment variable is not recognized; all character comparison is done in the C language locale. (See **environ**(5)).
- Nsort always performs a "stable" sort. Nsort always outputs records with the same key value in the same relative order in which those records appear in the input; POSIX sort does not guarantee this.
- If no POSIX arguments are given to Nsort other than the input file(s) and output file, Nsort will perform a sort using entire records as the default key. This is nearly identical to POSIX sort except that Nsort will pad comparisons between records of unequal length using the ASCII space character, whereas POSIX sort will consider the shorter of two otherwise equal records as having the lower sort value. The **-posix** command line argument will direct Nsort to follow the POSIX semantics for default key comparisons. The only case when the **-posix** argrument is needed is when no other POSIX command line arguments are given other than input and output files.

Windows Sort Compatible Command Line

The Windows version of Nsort accepts command line sort specifications for the Microsoft Windows sort command. The C locale must however be specified on the Nsort command line:

nsort /L[OCALE] C [**/R**] [*/+n*] [**/M** *kilobytes*]
[**/RE[CORD_BYTES**] *kbytes*] [**/CA[SE_SENSITIVE**]]
[[*drive1*:][*path1*]*filename1*] [/T [*drive2*:][*path2*]]
[**/O** [*drive3*:][*path3*]*filename3*]

/**L[OCALE] C** This ``option´´ is currently required for Nsort. That is, the C locale is the only locale option and must be explicitly specified in order to specify a key with the Windows sort notation. By default, the sort is case insensitive.

/+n Indicates the character number, n, of the beginning of the sort key in each record. For instance, /+2 specifies that comparisons should start on the 2nd character in each line. Comparisons start on first character of each line by default.

/**CA[SE_SENSITIVE]** Specifies that Nsort should use case-sensitive key comparisons.

/**M[EMORY]** *n* Indicates the number of kilobytes of main memory to use. The minimum memory size is 8000 kilobytes.

/**REC[ORD_MAXIMUM]** *n* Indicates the maximum number of characters per input line (default 4096, maximum 65535).

/**R[EVERSE]** Reverses the sort order; i.e. Z to A and 9 to 0.

[*drive1*:][*path1*]*filename1* Specifies an input file to be sorted. There can be multiple input files. If none are specified, the standard input is read.

/**T**[**EMPORARY**] [*drive2*:][*path2*] Indicates the path of a directory to hold the Nsort temporary file. For better temporary file performance, multiple temporary file paths can be specified on separate physical disks. The system temporary directory is used if no temporary directory is specified.

/**O**[**UTPUT**] [*drive3*:][*path3*]*filename3* Specifies the file where the sorted input is to be stored. If not specified, the data is written to the standard output. Specifying the output file is faster than redirecting standard output to the same file.

Nsort and Windows Sort Differences

Nsort differs from the standard Windows sort in the following ways:

- Nsort runs much faster for all but the smallest sorts.
- Nsort currently only performs Windows style sorts using the **C** locale, which must be specified on the command line as "/L C".
- Nsort always performs a "stable" sort. That is, Nsort always outputs records with the same key value in the same relative order in which those records appear in the input; Windows sort does not guarantee this.
- Nsort requires more memory to run than Windows sort.

Specification Language Overview 3

Nsort's specification language enables you to describe your sort data and sort operations, name the files to be used, and set configuration and performance options.

A sort definition statement consists of an Nsort command and any qualifiers associated with the command. Nsort commands specify a sort action or describe an element of the sort. The qualifiers modify the action of the command.

Sort definition statements can be specified on the command line, in sort specification files, or in Nsort user or system default files. Nsort commands and qualifiers are case-insensitive and can be abbreviated to the shortest unique string.

This chapter presents an overview of what must be specified for a sort and how it fits together. The following sections are included:

Nsort Statements

An Nsort statement that describes the keys, qualifiers, and other characteristics of a sort contains:

- A statement start character: either a dash (`-`) or a slash (`/`).
- The statement keyword name.
- A value containing the qualifiers for this statement (optional).

A command that is not merely a boolean option needs a command value. The value starts with either an equal sign (`=`) or a colon (`:`) and is followed by a constant, expression, or a list of qualifiers. A qualifier has a keyword and optionally its own value.

```
# Some simple statements
/field: name=total, decimal, offset=10, size:10
-statistics
-summarize=total
/field: name=balance, decimal, offset=0, size:10
/omit: balance < 0
```

Nsort's handling of keywords is flexible:

- Keywords are recognized in any mixture of upper and lower case.
- Keywords can be abbreviated by their first few letters, as long as it is distinguishable it from the other keywords permissible in that context. Two or three letters are usually enough.

  ```
  # These examples are equivalent:
  -key:offset=0, size=4, binary
  -ke:o:0, si=4, bin
  /key=bi, si=4, of=0
  ```

- Keywords containing more than one English words contain optional underscores as word separators, as in `file_system`. Nsort recognizes these long keywords whether or not the underscores are included.

The statement start character (`-` or `/`) and its keyword must be adjacent; elsewhere blanks, tabs and newlines may be added for clarity. Comments start with either a number sign (`#`) or an exclamation point (`!`) followed by a blank and end at the next newline character.

Character Constants

Records and fields have some properties that are single character values, for example, the terminating character of a delimited record. Nsort has built-in identifiers for some of these characters:

Nsort Identifier	Character
backslash	Backslash (\).
blank	Space or blank ( ).
colon	Colon (:).
comma	Comma (,).
cr	Carriage return.
dollar	Dollar sign ($).
dquote	Double quote (").
newline or **nl**	Newline (\n).
null	Null (\0).
pipe	Vertical bar or pipe (|).
semicolon	Semicolon
space	Space or blank ( ).
squote	Single quote (').
tab	Tab (\t).

You can also specify a character by enclosing it in single quotes ('), or using one of these escape sequences:

- '\n': the newline character.
- '\t': the tab character.
- '\0nnn': the character with the octal value 0nnn.
- '\\' the backslash character (\).
- '\'' the single quote character (').

Processing Nsort Commands

Nsort builds a sort definition from several sources. It first reads any system-wide default statements from the Nsort system-wide default parameter file if that file exists and is readable. You can store the tuning parameters for your system here.

```
# This temp_file statement could help the speed of sorts
# on a system which has four temporary file systems available
-temp=/tmp1, /tmp2, /tmp3, /tmp4
```

Nsort next looks in the *.nsortrc* file in your home directory for your personal default statements.

Nsort then reads any statements in the NSORT environment variable.

Lastly, Nsort reads the statements on the command line, processing them from left to right. Any specification files given on the command line are processed in order.

Any sort definition statement can appear in any of the above locations. POSIX options are only recognized on the command line.

If there is a conflict between options specified in different sources, the last-read option take precedence over the earlier options. An option specified in the system-wide defaults can be overridden in your user defaults. Options specified on the command-line or in a specification file take precedence over the default options.

Refer to "Global Options" on page 21 for more information on setting global options for Nsort.

Data Definition Statements

Data definition statements describe the records, fields and keys used in the sort. The data definition statements are as follows:

Nsort Command	Description
format	Specifies the record definition format of the input file.
field	Describes important fields in the input file.
key	Denotes the key fields in each record.

This section presents an overview of data definitions statements and covers the following topics:

- Record definition (page 34).
- Field definition (page 35).
- Key definition (page 36).

For a detailed presentation of Nsort data definitions, see Chapter 4, "Describing the Sort Data".

Record Definition Qualifiers

Record definition qualifiers modify the behavior of the **format** command. Record definition qualifiers describe the type of records (fixed-length, variable or delimited) and any qualifiers that apply to the records. The record definition qualifiers are as follows:

Nsort Command	Description
size:*Number*	Tells Nsort that the input data is organized as fixed-length records and specifies the length of each record as *Number*.
size:variable	Tells Nsorthat the input data consists of variable length records that are preceded by a 2-byte unsigned integer that indicates the number of remaining bytes in the record.
delimiter:*C*	Tells Nsort that the input data is organized as records delimited by the character *C*.
separator:*C*	Indicates that record fields are separated by the character *C*.
skip_blanks	Tells Nsort to skip over blanks and tab characters when comparing record keys.
minimum_size	Specifies the minimum expected length for delimited records.
maximum_size	Specifies the maximum expected length for delimited records.

For a detailed discussion of the **format** specifier, see "Record Formats" on page 46.

Field Definition Qualifiers

Field definition qualifiers describe the individual fields in the records. The field definition qualifiers are as follows:

Nsort Command	Description
name	Provides the field with a convenient label that can be used later in other Nsort commands.
position	Specifies the absolute position of the field in bytes (starting from 1).
offset	Specifies the absolute offset of the field in bytes (starting from 0).
size	Specifies the exact length of the field in bytes.
delimiter	Specifies the delimiter at the end of the field.
pad	Character with which to pad the field for comparisons (default ASCII blank).
maximum_size	Specifies the maximum number of characters for the field.
data_type	Specifies whether the field contains text, a month, a binary integer, a packed decimal, ascii decimal, or floating point number.

The following example describes the `sales` and `region` fields:

```
-field=sales, binary, size:8,
       region, char, size:20
```

For a detailed discussion of field definitions, see "Field Definitions" on page 52.

Key Definition Qualifiers

Key definition qualifiers describe the key fields upon which to base the sort. The key definition qualifiers are as follows:

Nsort Command	Description
name	Provides a convenient label for the key or key field.
ascending	Tells Nsort to perform an ascending (lowest-to-highest) sort.
descending	Tells Nsort to perform a descending (highest-to-lowest) sort.
number	Specifies the significance of the key in the sort.
position	Specifies the absolute position of the key field in bytes (starting from 1).
offset	Specifies the absolute offset of the key field in bytes (starting from 0).
size	Specifies the exact length of the key field in bytes.
delimiter	Specifies the delimiter at the end of the key field.
pad	Character with which to pad the key field for comparisons (default ASCII blank).
data_type	Specifies whether the field contains text, a month, a binary integer, packed decimal, ascii decimal, or floating point number.

The following example orders records by the `sales` field in descending order and secondarily by the `region` field in ascending order:

```
-field=sales, decimal,
       region
-key=region
-key=sales, descend, number:1
```

For a detailed discussion, see "Key Definitions" on page 64.

Supported Key and Field Data Types

Nsort supports the following data types for field and key data:

Nsort Command	Description
integer **binary**	1, 2, 4, or 8 byte integer.
unsigned **unsigned integer** **unsigned binary**	1, 2, 4, or 8 byte unsigned integer.
packed	Packed binary coded decimal.
character	Fixed length or delimited ASCII character string.
float	32-bit IEEE floating point number.
double	64-bit (double-precision) IEEE floating point number.
decimal	ASCII character string representation of a decimal number.
month	Three-letter character string specifying a month of the year.

For a detailed discussion of Nsort data types, see "Data Types" on page 67.

Sort Definition Statements

Sort definition statements specify the basic commands for Nsort. The sort definition statements specify when to perform a merge instead of a sort, whether to delete duplicates or summarize data, which data to include or omit in the sort, and how to lay out the records.

By default, Nsort sorts the standard input as lines of text and writes the results to the standard output without deleting records with duplicate key values.

The top level sort commands are as follows:

Nsort Command	Description
merge	Merges several sorted input files into a single output stream.
summarize	Sums the values of the specified fields to produce summary records by key value.

The following example uses the **summarize** command to produce a summary file containing sales by region:

```
# Generate a summary file of sales by region.
-format:size=8
-field=region, binary, size:4,
       sales, binary, size:4
-key=region
-summarize=sales
```

The top-level sort options modify the sort or merge command:

Nsort Command	Description
duplicates	Includes records with duplicate key values in the output stream. On by default.
no_duplicates	Omits records with duplicate key values.
include	Selects records with the specified criteria to be included in the sort.
omit	Omits records with specified criteria.
derived	Derives a new field from the values of existing fields.
reformat	Lays out the record format for the sort.
count	Sorts at most the specified number of input records.
condition	Define the named conditional expression.
specification	Read commands from specification file.
add_derived	Derived fields are automatically added to records. On by default.
no_add_derived	Derived fields are not automatically added to records.
warnings	Allow warning messages indicating that unusual, non-fatal conditions have occurred, e.g. overflow of a summary field or the detection of excessive paging. Nsort continues with the operation. On by default.
no_warnings	Disable warning messages.

For a detailed presentation of Nsort sort options, see Chapter 5, “Sort Definition Statements”.

File Definition Statements

The file definition statements specify the filenames of the input, output, sort specification, and temporary files for Nsort to use during the sort. The file definition statements are as follows:

Nsort Command	Description
in_file	Specifies an input file. If absent, Nsort uses the standard input. You can specify multiple input files.
out_file	Specifies an output file. If absent, Nsort uses the standard output. You can specify multiple output files.
temp_file	Specifies a temporary file for use by the sort. You can specify multiple temporary files.
file_system	Specifies default settings for a specific file system.

For a detailed presentation of file definition statements, see "File System Defaults" on page 105.

File Qualifiers

The file qualifiers specify the I/O options for a given file. The file specification qualifiers are as follows.

Nsort Command	Description
direct	Tells Nsort to use direct I/O for file access.
mapped	Tells Nsort to use memory-mapped I/O for file access.
buffered	Tells Nsort to use buffered I/O for file access.
transfer_size	Specifies the file transfer size.
count	Specifies the maximum number of asynchronous I/O requests.

For a detailed presentation of file I/O qualifiers, see "File I/O" on page 104.

Configuration and Performance Statements

The configuration and performance statements set memory, processor, sort method and statistics options for a sort. The configuration and performance statements are as follow:

Nsort Command	Description
memory	Specifies how much memory Nsort should use.
threads ***or*** **processes**	Specifies how many internal threads Nsort should use for the bulk of the sort workload.
method	Tells Nsort which sort strategy to use.
statistics	Tells Nsort to provide statistics on the sort.
no_statistics	Tells Nsort not to provide statistics on the sort. On by default.

Refer to Chapter 6, “Configuration and Performance” for more information on configuration and performance commands.

Method Qualifiers

The sort method qualifiers tell Nsort to use the specified sort strategy. The sort method qualifiers are as follows:

Nsort Command	Description
record	Tells Nsort to move complete records as it sorts.
pointer	Tells Nsort to move only keys as it sorts, using pointers to locate the original records.
merge	Tells Nsort to use a merging sort strategy.
radix	Tells Nsort to use a radix sort strategy.
no_radix	Tells Nsort not to use a radix sort strategy. On by default.
hashing	Tells Nsort to use key hashing in its sort.
no_hashing	Tells Nsort not to use key hashing in its sort. On by default.

Refer to "Sort Methods" on page 100 for more information on sort methods.

Describing the Sort Data 4

Nsort data description statements define:

- The record type, including size or delimiter.
- The types and locations of any fields that you use as keys or in expressions or conditions.
- The order of key fields and the direction of the sort.

This section discusses Nsort's data description statements and includes the following sections:

Record Formats

The **-format** statement defines whether a record is fixed-size, variable (length-prefix) or delimited by a character constant. For delimited records, you can also specify field separators, and minimum and maximum record sizes.

-format= size:{*number* | **variable**} |
{[**delimiter:***C*] [,**separator:**{*C* | **whitespace**}]
[,**default**:{*C* | *number* | *character_string*}] [,**skip_blanks**]
[,**minimum_size:***number*] [,**maximum_size:***number*]}

The default record format is lines of text delimited by a newline, with the fields separated by whitespace (blanks and tabs).

This section discusses record formats and includes the following subsections:

Fixed Size Records

Fixed-size records are all the same size. To specify fixed-size records, use the **size:***number* qualifier.

```
-format=size:30 # Each is 30 bytes long
```

Figure 4-1 Fixed Size Records

```
John Smith           M10/03/38
Mary Jones           F03/09/68
William Evert        M05/20/58
Jennifer Small       F08/19/42
Ben Liu              M06/23/68
Alfred Hollingsworth M02/25/52
Jackie Jacobs        F12/24/75
Susan Hollings       F05/29/51
            .
            .
            .
```

All fields in a fixed-size record must be fixed-size. Only fixed-size records may contain binary integer, packed decimal and floating-point fields.

The maximum size of a fixed-size record is 65,535 bytes (or 8 megabytes for the Windows versions of Nsort).

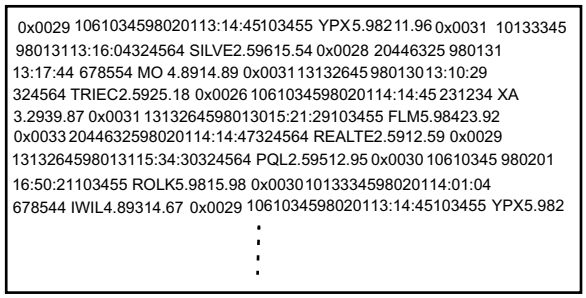
Variable (Length-Prefix) Records

Variable, length-prefix records are records of varying lengths with a two-byte size at the beginning of each record. To specify variable-length records, use the **size:variable** qualifier.

```
-format=size:variable
```

The size field is an unsigned integer that may range from 0 to 65,535. It is treated as a prefix to the record rather than as part of the record; therefore it is not included in the record length nor can it be accessed as a field or key. The total record size, including the size field, varies from 2 to 65,537 bytes.

Figure 4-2 Variable Length Records



The following functionality is not supported with variable (length-prefix) records:

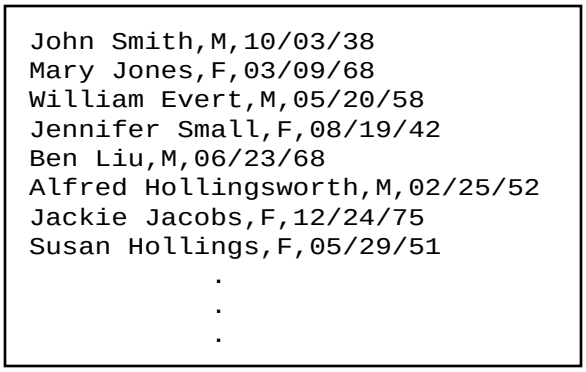
- record selection
- reformatting
- derived fields

Delimited Records

Delimited records can vary in size and are terminated by a single character. To specify delimited records, use the **delimiter:***character* qualifier. Two examples follow:

```
-format=delimiter:newline # lines of text, the default format
-format=delimiter:null    # ASCII NULL terminated “lines”
```

Figure 4-3 Delimited Records



The default maximum size for a record is 4,096 bytes. The maximum record size allowed can be increased up to 65,535 bytes.

Field Separators

The **separator** qualifier can be used to specify the field separator character.

```
-format=separator:comma  # lines of text (by default) with
                         # fields separated by commas
```

A field separator can either be whitespace (a maximal sequence of adjacent blanks or tabs) or a single character.

Whitespace separated fields include their preceding whitespace; any trailing whitespace is part of the subsequent field. Whitespace is the default field separator for delimited records.

A single-character separator is not included in the field. Two adjacent separator characters, or a separator character at the beginning or ending of the record denote empty fields.

Default Field Value

A default field value can be specified using the **default** qualifier. This default is only used in the case that:

- the fields are whitespace separated
- a reformat statement is specified
- and some of the fields specified in the reformat are empty. (For instance, there are not that many fields in the record, or the beginning and ending positions of the field are defined in such a way that the field is empty.)

If the above conditions are met, empty fields cannot be reliably represented with whitespace separators (since these separators can consist of multiple blank and tab characters). In this case, the character string specified by the **default** qualifier will be substituted for the non-existent field in the reformatted record. In the absence of a **default** qualifier, the string "*NULL*" will be used.

```
-format:default="*Error*"  # Use *Error* as default field value
                           # for reformats
```

Skipping Blanks

The **skip_blanks** qualifier specifies that the starting and ending positions of separated fields begin after any leading blanks that would otherwise be included in those fields.

Minimum and Maximum Record Sizes

You can specify minimum and maximum sizes for delimited records using the **minimum_size:***number* and **maximum_size:***number* qualifiers. If you provide conservative approximations of these values, Nsort can more accurately calculate its memory needs. Note that a run-time error will occur if Nsort encounters a record that is smaller than the minimum or larger than the maximum.

The following example specifies newline-delimited records between 11 and 80 bytes long:

```
-format=delim:nl, minimum_size:11, maximum_size:80
```

The following example specifies a NULL-delimited records with a minimum size of 20 bytes:

```
-format=delim:null, min:20
```

The default minimum size is the position of the last byte of all declared fixed-sized fields, plus one byte for the record delimiter. The default maximum size is 4,096 bytes. The maximum can be increased up to 65,535 bytes.

The following example specifies newline-delimited records with a maximum record size of 1000 bytes. Nsort will use a minimum record size of 22 bytes, based on the sizes of the specified fields:

```
-format=delim:nl, max=1000
-field= city, position=12, size=10,
        state, position=2, size=3
```

Field Definitions

Each field statement identifies one or more record fields for later use in expressions, keys, summaries, and reformats. To specify a field, describe its position, size, and type information.

-field= [**name=**]*field_name* [,{**position** | **offset**}**:** *number*]
[**,** {**size:***number* | **delimiter:***character* }] [**,** *datatype*] [**, pad:***character*]
[,{**maximum_size** | **minimum_size**}**:***number*] [, ...]

Nsort supports three different types of fields: fixed-size; delimited and separated. The following table illustrates how the beginning and ending positions for these different field types are defined.

Field Type	Beginning Position	Ending Position
fixed-size	absolute byte position from beginning of record	implicitly from the fixed size of the field
delimited	absolute byte position from beginning of record	delimited by record delimiter or field separator
separated	relative to a field separator	relative to a field separator

Field statements are additive. Multiple fields can be defined in one or more field statements. For instance, the following definition of three separated fields:

```
-field=first,second,third
```

is equivalent to the following three field statements:

```
-field=first  -field=second  -field=third
```

This section discusses field definitions. The following qualifiers are presented:

- Name (below).
- Size (below).
- Delimiter (next page).
- Position (page 55).
- Offset (page 63).
- Maximum size (page 63).
- Pad (page 63).

Field Name

Fields must be specified with a name qualifier. The name can be used in subsequent expressions or key or summarize statements. Field names start with a letter, and can contain letters, digits, and underscores (_).

[**name=**]*field_name*

The **name**= prefix is optional, and is useful for specifying field names that are also keywords in the Nsort specification language. E.g.:

```
# Define fields “name”, “height”, and “size”
-field= name=name, size=24,
        height, decimal, size:4,
        name=size, decimal, size:2
-key= name=size
```

Size Qualifier

Fixed-size fields are specified with a **size** qualifier.

size:*number*

The specified size must be greater than 0.

Any **position** or **offset** qualifier for a fixed-size field indicates the byte position of the field relative to the beginning of the record. All fields in a fixed-size record must be fixed-size.

Separated fields are the norm for delimited records. Fixed-size fields can also be defined for delimited records with following caveats:

- A run-time error will result if the record is not large enough to contain all fixed-size fields.
- Fixed-size fields in a delimited record may not appear in a record reformat.

Delimiter Qualifier

A delimited field begins at a fixed byte position in the record and ends at the first occurrence of *character* or at the end of the record.

delimiter:*character*

The delimiter is not included in the field. A field is empty if it starts with the delimiter. A run-time error will result if a delimited record is not large enough to contain the first byte of a delimited field.

Delimited fields cannot be included in a record reformat, and are not supported for fixed-size records.

Fields which start after a field separator should be defined as separated fields using the **position** qualifier instead of the **delimiter** qualifier.

Position Qualifier

The position qualifier defines the beginning of the field. The meaning of the position number depends on whether a **size** or **delimiter** qualifier is specified for the field.

Byte-Position Fields

If a **size** or **delimiter** qualifier is specified for the field, the position *number* specifies the byte number in the record.

position:*number*

A field at the beginning of the record has a position of 1. The position cannot be greater than the size of a record. A run-time error will occur if a delimited record is not large enough to contain a byte-position field.

In the following example defines a 4-byte unsigned binary number beginning at the fifth byte of the record:

```
-format=size:12
-field=unix_date,position:5,binary,unsigned,size:4
```

If the position is not specified for a field, it is assumed to immediately follow the previously defined field. Thus if all fields in a record are defined in order, the positions of the fields need not be explicitly specified. The following example shows position-implicit field specifications for a fixed-size record.

```
-format=size:12
-field=ip_addr,binary,unsigned,size:4,
       unix_date,binary,unsigned,size:4,
       page_id,binary,unsigned,size:4
```

Separated Fields

The position *qualifier* for a separated field specifies the field's starting and ending range. (The field separator character can be defined in a **format** statement.) In addition to the field number, an optional field end and field type can be specified:

position: *field_no*[.*char_no*][**bdfiMnr**][-[*field_no*[.*char_no*][**b**]]]

Examples:

```
-field=amount,pos=1n   # “amount”, 1st field, and is ascii number
-field=store,pos=3     # “store”, 3rd field, is character string
```

The beginning and ending positions (and any modifiers) in a **position** qualifier cannot contain any whitespace characters (i.e. they must be specified contiguously). This is unlike other parts of the Nsort sort specification language where space or tab characters may appear between keywords.

Separated Fields vs. Input Fields

The **position** qualifier can be used to define a separated field that is either a whole field (separated by field separators) in each input record, a portion of an input record field, or multiple input record fields. Hence the term *input fields* will be used here to refer to the fields separated by field separators in the input records, and *separated fields* as the potential input field fragments or groups that are defined with a **field** statement and **position** (or **offset)** qualifier.

Beginning Field Number

The beginning field number used by itself defines whole input fields. The following example defines the third, fifth, and sixth fields separated by the pipe ('|') character:

```
-format=separator:pipe
-field=street,position:3,
       state,position:5,
       zip_code,position:6
```

If the position is not specified for a field, it is assumed to immediately follow the previously defined field. The positions of fields need not be explicitly specified if all fields in a record are defined in order. The following example shows position-implicit, separated fields:

```
-format=separator:pipe
-field=customer_name,
       telephone_no,
       street,
       city,
       state,
       zip_code
```

The *char_no* can be used to specify the beginning character of the separated field.

```
-field=account_no,position:1.2   # starts at 2nd character of
                                 # 1st field
-field=balance,position:4.3      # starts at 3rd character of
                                 # 4th field
```

If the *char_no* is past the end of the input field specified by *field_no*, the separated field will begin at the field separator or record delimiter which follows the input field. Depending on the ending field definition, this may cause the separated field to be empty.

Following the tradition of POSIX sort, whitespace-separated fields include any preceding whitespace characters by default. This behavior can be overridden by specifying **skip_blanks** in the **format** statement. For all other separators, the first character of a field is the first character after the preceding separator character or, for the first field, the first character of the record.

If the **b** modifier is used with a *char_no* or if the **skip_blanks** qualifier has been specified in the **format** statement, the *char_no* position is relative to the first non-blank character in the field. Two examples follow:

```
-field=client_name,position:3.2b # starts at 2nd character after
                                 # 1st non-blank character in
                                 # the 3rd field

-format=skip_blanks
-field=client_name,position:3.2  # starts at 2nd character after
                                 # 1st non-blank character in
                                 # the 3rd field
```

Ending Field Number

The ending field number is optional. In its absence the end of the separated field is the end of the beginning field number. An ending *field_no* without an associated *char_no* defines the field as ending at the end of input field *field_no*. For example, the following field definitions are equivalent:

```
-field=customer,position:3.2
-field=customer,position:3.2-3
```

If an ending *char_no* is specified, it indicates the last character of the separated field. That is, the end character is included in the field rather than being the first character after the end of the field. For example:

```
-field=zip,position:4.1-4.5   # first 5 chars of 4th input field
-field=state,position:3.1-3.2 # first 2 chars of 3rd input field
```

If the **b** modifier is used with the ending position (or **skip_blanks** is specified in the **format** statement) the ending *char_no* is relative to the first non-blank character of the ending field.

If the **b** modifier is used with the ending position (or **skip_blanks** is specified in the **format** statement) and an ending *char_no* is not specified, then:

- if whitespace field separators are used, the **b** modifier is ignored
- if a single character field separator is used, the **b** modifier (or **skip_blanks**) causes blanks at the end of the input field to be dropped from the separated field

Open-ended Fields

Open-ended separated fields can be specified just using an ending hyphen:

```
-field=position:3-    # 3rd field and up to record delimiter
```

Open-ended fields may only be specified in a **reformat** statement as the last field.

Examples

Given the following record with whitespace-separated fields:

```
red  blue tan pink
```

the following field directives will result in the following separated field contents for the above record:

Field Directive	Contents	Comments
`-field=col2,position:2`	`“  blue”`	whitespace separated fields include beginning whitespace by default
`-format:skip_blanks` `-field=col2,position:2`	`“blue”`	
`-field=col2,position:2b`	`“blue”`	
`-field=col2,position:2.1-2.2`	`“  ”`	whitespace separated fields include beginning whitespace by default
`-format:skip_blanks` `-field=col2,position:2.1-2.2`	`“bl”`	
`-field=col2,position:2.1b-2.2b`	`“bl”`	
`-field=col2,position:2.1b-2.2`	`“”`	ending position is before beginning position, therefore the separated field is empty
`-field=col2,position:1-2`	`“red  blue”`	1st through 2nd fields
`-field=col2,position:1b-2`	`“red  blue”`	blanks not skipped between input fields
`-field=col2,position:3b-`	`“tan pink”`	third input field and up to record delimiter

Given the following record with pipe-separated fields (declared with `-format=separator:pipe`):

```
red| blue |tan|pink
```

the following field directives will result in the following separated field contents for the above record:

Field Directive	Contents	Comments
`-field=col2,position:2`	`" blue "`	
`-format:skip_blanks` `-field=col2,position:2`	`"blue"`	blanks skipped at beginning and end
`-field=col2,position:2b-2b`	`"blue"`	blanks skipped at beginning and end
`-field=col2,position:2.1-2.2`	`" b"`	
`-format:skip_blanks` `-field=col2,position:2.1-2.2`	`"bl"`	
`-field=col2,position:2.1b-2.2b`	`"bl"`	
`-field=col2,position:2.1b-2.2`	`"b"`	ending position is same as beginning position, therefore the separated field is one character
`-field=col2,position:1-2`	`"red| blue "`	1st through 2nd fields, separators not skipped
`-format:skip_blanks` `-field=col2,position:1-2`	`"red| blue"`	trailing blank is skipped
`-field=col2,position:3-`	`"tan|pink"`	third input field and beyond
`-field=col2,position:1.6-2`	`"| blue "`	beginning *char_no* bumps up against field separator

POSIX-Style Key Types

Many POSIX-style modifiers can be used to specify how the separated field will be interpreted in a key comparison. Unlike the **b** modifier, these modifiers do not change where the field starts or ends.

Modifier	Description
d	Use "dictionary" order. All characters except letters, digits, tabs and blanks are ignored.
f	Fold lower case characters into the equivalent upper case character. E.g. 'z' will be sorted the same as 'Z'.
i	Ignores characters less than ASCII 040 (octal) or greater than 0176.
M	The field contains a month name for comparison purposes. Any leading white space is ignored. If the field begins with the first three characters of a month in uppercase or lowercase, comparisons are made according to month order. Any invalid month names are compared as less than JAN.
n	The field is a number consisting of optional blanks, an optional '+' or '-' sign, zero or more digits, an optional decimal point, and zero or more digits. An exponent may also be included with "**n**" separated fields (e.g. 3.76e10). Note that exponents are not recognized with POSIX sort keys (defined on the Nsort command line with -k, and +w.z -y.z).
r	Reverses the order of comparison so that keys appear in the output in descending order.

An example follows:

```
-field=client,position:3fr  # client is 3rd field
-key=client      # reverse order, ignore case
```

Offset Qualifier

The **offset** qualifier is the same as the **position** qualifier, except that the first byte, field, or character is always specified as 0, rather than 1. A position of N is the same as an offset of N-1. For example, the following two field definitions are the same:

```
-field=zip,position:4.1-4.5
-field=zip,offset:3.0-3.4
```

Maximum Field Size

The **maximum_size:***number* qualifier specifies maximum number of characters in a separated field.

```
# “cost” field is ascii number with no more than 10 characters
-field=cost,decimal,maximum_size:10
```

This qualifier can be used to specify the maximum precision of a separated field that is the target of a **summarize** statement (see “Summarizing Separated Fields” on page 78).

Pad Qualifier

Nsort uses the pad character when comparing character strings of different sizes. The comparison is made as if the shorter string were appended with the pad character to form a string of equal size. The pad qualifier changes the pad character from ASCII blank to the value *character*. The pad character is ignored for non-character data types.

pad:*character*

Key Definitions

The key definition describes the key fields used in each sort, the direction of the sort on each key field, and the relative significance of each key field. If you do not define any key fields, the entire record is sorted as a single character string in ascending order.

This section discusses key definition statements and includes the following subsections:

- Key fields in Nsort.
- Key sort direction.
- Key number.

Key Fields

Nsort supports two types of key descriptions, named keys and described keys. This section discusses key descriptions and includes the following subsections:

- Named keys.
- Described keys.

Named Keys

Nsort keys are fields with two optional properties as follows:

- An indicator of whether the key is to be sorted ascending or descending.
- A number specifying the relative ordering priority among multiple keys.

-key= [**name=**] *field_name* [**,** {**ascending** | **descending**}] [**, number:***number*]

Nsort obtains this key field's placement, size, and other information from the previously defined field *field_name*. The optional **name=** qualifier may be used to eliminate the ambiguity that arises when *field_name* matches the first letters of a potential keyword in a **-key** statement, as in the following example:

```
# "name=" is necessary here to distinguish
# the field named "mon" from the type "month"
-key=name:mon
```

Described Keys

Instead of using named key fields, you can roll the field definition into the key definition as follows:

-key= [{**position** | **offset**}**:** *number*] [**,** {**size:***number* | **delimiter:***character*}]
[**, datatype**] [**, pad:***character*] [**,** {**ascending** | **descending**}]
[**, number:***number*] [, ...]

Refer to "Field Definitions" on page 52 for information on describing key fields. A described key example follows:

```
-key=position:2nr  # first sort on 2nd input field as a
                   # decimal number in reverse order,
-key=position:4f   # then 4th field as string ignoring case,
-key=position:5-6  # then 5th and 6th fields
```

Key Sort Direction

By default Nsort performs an **ascending** sort (orders a key field from low values to high ones). You can indicate **descending** to reverse the sort order of a key; Nsort then sorts this key field from high values to low ones.

In the following example records are ordered by the second separated field in descending order:

```
-key=position:2,descending
```

Key Number

By default, the first key field defined is the most significant; subsequent key fields are used only when the previous key fields are equal. You can use **number:***number* qualifiers to override the default behavior. The key field with the lowest number value is the most significant, the one with the second lowest is next, and so on. Any remaining key fields are used in order of their definition.

The following example orders records primarily by the `region` field in ascending order, and secondarily by the `sales` field in descending order:

```
-field=sales, decimal,
       region
-key=region
-key=sales, descend
```

The following example orders records primarily by the `sales` field in descending order, and secondarily by the `region` field in ascending order:

```
! Order records primarily by the sales field in descending
! order, and secondarily by the region field in ascending order.
-field=sales, decimal,
       region
-key=region
-key=sales, descend, number:1
```

Data Types

Nsort supports the following numeric and character data types:

- Signed integer (**integer** or **binary**).
- Unsigned integer (**unsigned integer** or **unsigned binary**).
- Packed decimal.
- Character (**character**).
- Floating point (**float**).
- Double-precision floating point (**double**).
- Decimal (**decimal**).
- Month (**month**).
- Unsigned bit field (**bit**).

Binary Integer Data Types

The **integer** and **binary** types denote a two's-complement number that is 1, 2, 4, or 8 bytes long. It is a signed number unless you include the **unsigned** modifier.

```
-format=size:16
# sales: an 8-byte signed integer at offset 4
-field=sales, bin, offset:4, size:8
# salary: a 4-byte unsigned integer at offset 12
-field=salary, unsigned, bin, off:12, siz:4
```

An **integer** field is only allowed with fixed-size or length-prefix records.

Packed Decimal Type

The **packed** type denotes a packed binary coded decimal number. Each digit is represented in a 4-bit nibble with a value from 0 to 9. The **size** is the number of nibble-pairs or bytes in the field. The last nibble contains a sign indicator with one of the following hexadecimal values: A (+); B (-); C (+); D (-); E (+); or F (absolute, no sign).

```
-format=size:16
# sales: a 7-digit (4-byte) packed decimal number at offset 2
-field=sales, packed, offset:2, size:4
```

The **packed** field is only allowed with fixed-size or length-prefix records.

Bit Type

The **bit** type is a sequence of 1 to 8 bits, all contained in a single byte. **Bit** types are always unsigned and are only allowed with fixed-size or length-prefix records. The **position** or **offset** of a bit type indicates the position of the byte containing the bit type, plus the bit number of the high-order bit of the field. The **size** option can be used to specify bit type greater than one bit in length.

```
-format=size:16
-field=flag1,position:5.1,bit  # low-order bit of fifth byte
-field=flag2,position:5.8,size=4,bit # 4 high-order bits, fifth byte
```

Character Data Type

A **character** string is an unstructured sequence of bytes of any size from 1 to the size of the record. Strings can be of a fixed or varying size. When two character strings of different sizes are compared, the result is as if the shorter string were filled out with the pad character (default: ASCII blank) until it has the same size as the longer string. The pad may be changed from ASCII blank to any other character by using the pad:*character* qualifier.

```
! employee: a 20-byte character field at position 11
-field=employee, char, pos:11, size:20

! street: a null-terminated character string at offset 2
-field=street, char, off:2, del:null
```

Floating Point Data Type

A **float** is a 32-bit IEEE 754 data type. It always has a size of 4, and therefore need not have an explicit size specification.

```
-format=size:14
# Xcoordinate: a 4-byte floating point at offset 10
-field=Xcoordinate, offset:10, float
```

The **float** data type is only allowed with fixed-size or length-prefix records.

Double-Precision Floating Point Data Type

A **double** is a 64-bit IEEE 754 data type. The **double** type always has a size of 8, and therefore need not have an explicit size specification.

```
-format=size:16
# sales: an 8-byte floating point at offset 4
-field=sales, offset:4, double
```

The **double** data type is only allowed with fixed-size or length-prefix records.

Decimal Data Type

A **decimal** is a character string containing an ASCII representation of a number in the form:

[**+** | **-**][*digits*][**.**][*digits*][{**E** | **e**}[+ | -][*digits*]]

The number can be preceded by spaces and is terminated by the end of the string or an unexpected (e.g. non-digit, decimal point) character. A string which is too short (e.g. "+", ".") or starts with unexpected characters (e.g. "+r") is treated as zero.

Month Data Type

A **month** contains a three letter abbreviation for a month name and is ordered according to the months of the year; e.g. `jan` or `Feb`. The **locale**(5) database holds the standard names. Case distinctions are ignored for month types. Month values whose first three letters do not match those abbreviations sort as less than January.

```
# mm: a month starting offset 4
-field=mm, offset:4, month, size:3
```

Sort Definition Statements 5

Nsort sorts, merges or summarizes. In sort mode, Nsort takes an input file and sorts it by key value. In merge mode, Nsort takes multiple pre-sorted input files and produces one sorted output file. In summarize mode, Nsort takes an input file and produces a summary file with selected values subtotaled by key value.

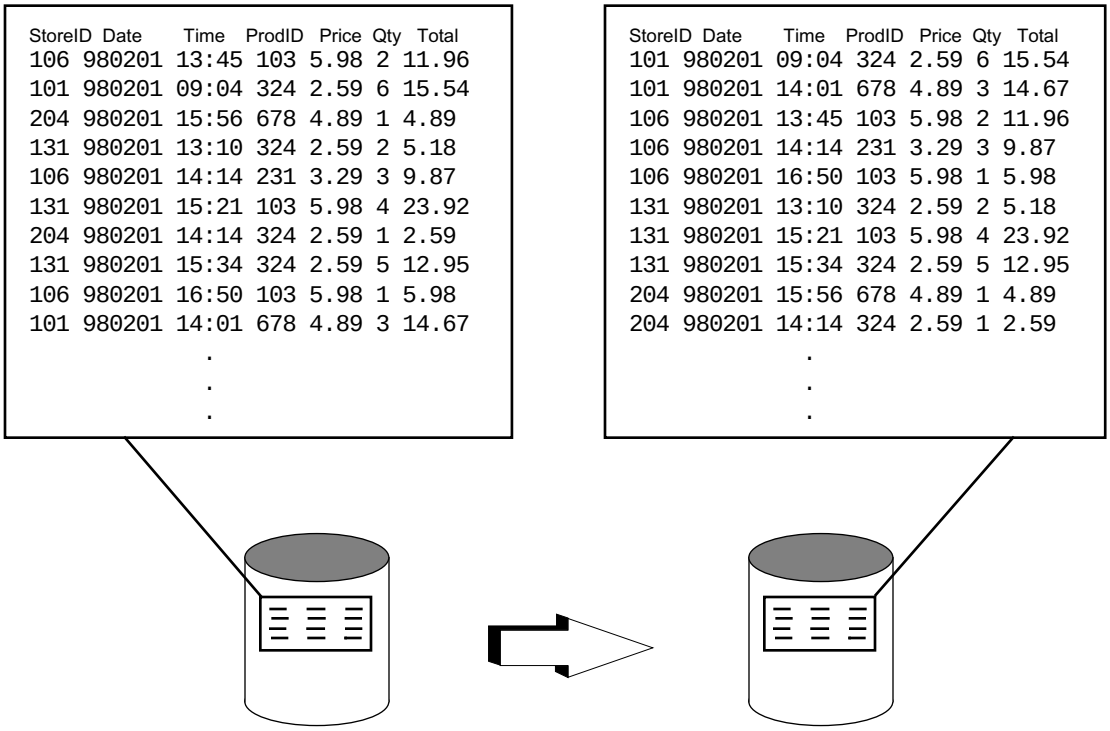
While Nsort works, it can transform records. You can rearrange or drop fields, add new fields with derived statements, or produce subtotals.

This chapter describes the basic sort operations provided by Nsort and includes the following sections:

Sorting

Sort takes one or more unordered input files and sorts them by text line or key value into one or more output files. Sort is the Nsort's default mode; in the absence of another mode specification (**merge** or **summarize**), Nsort sorts the input.

Figure 5-1 Sort Operation




```
-field=StoreID,Date,Time
-key=StoreID
-key=Date
-key=Time
```

Nsort provides many options to customize your sort, including the following:

- Record selection.
  You can select records to include or omit from the sort.
- Record reformat.
  Nsort provides the reformat option to project the input records into any form you choose. You can specify which fields to include in the sort and the order in which those fields appear.
- Duplicate key processing.
  You can select whether to delete or include records with duplicate key values in the sort. The default is to include them.
- Field summarizing.
  Field values can be subtotaled by key value.
- Derived fields.
  You can derive new fields from the values of existing fields.
- Derived fields.
  You can derive new fields from the values of existing fields.
- Key match detection.
  You can specify that a character be prepended to each output record that indicates whether the keys in the record match the same keys in the previous output record.

Merging Sorted Sets

Merge takes several sorted input files and merges them to produce one or more sorted output files. To specify merge mode, use the **merge** command.

-merge

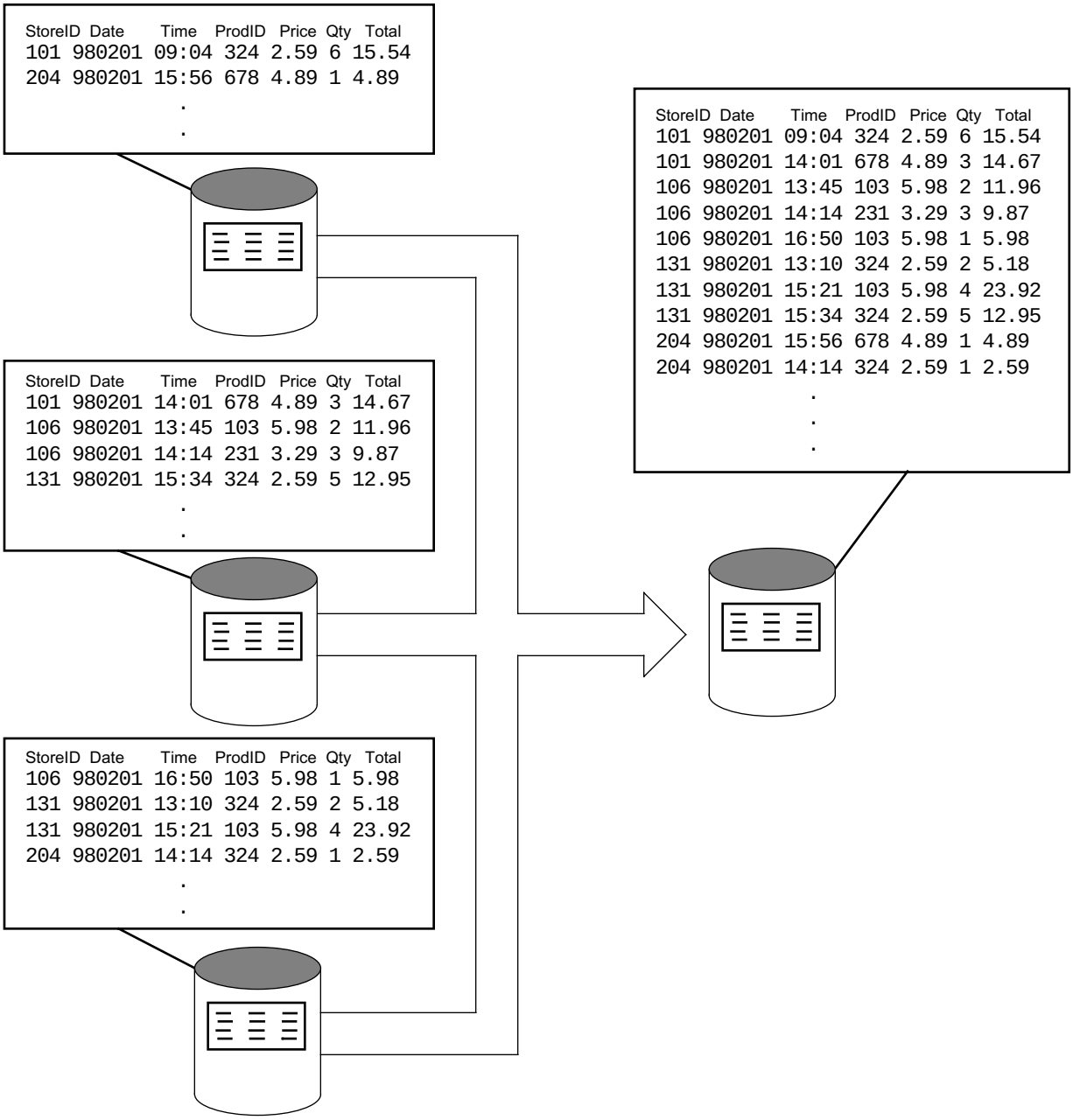
Nsort provides the following options to customize your file merging:

- Duplicate key processing.
  You can select whether to delete or include records with duplicate key values in the sort. The default is to include them.
- Field summarizing.
  Field values can be subtotaled by key value. (Field summarizing during a merge is currently supported only for fixed-size records.)
- Multiple output files.
  Multiple output files can be specified, each with its own record selection and reformat statements.

Figure 5-2 on page 75 shows how the merge operation combines sorted input files from the following command statements:

```
-merge
-field=StoreID,
       Date,
       Time
-key=StoreID -key=Date -key=Time
```

Figure 5-2 Merge Operation



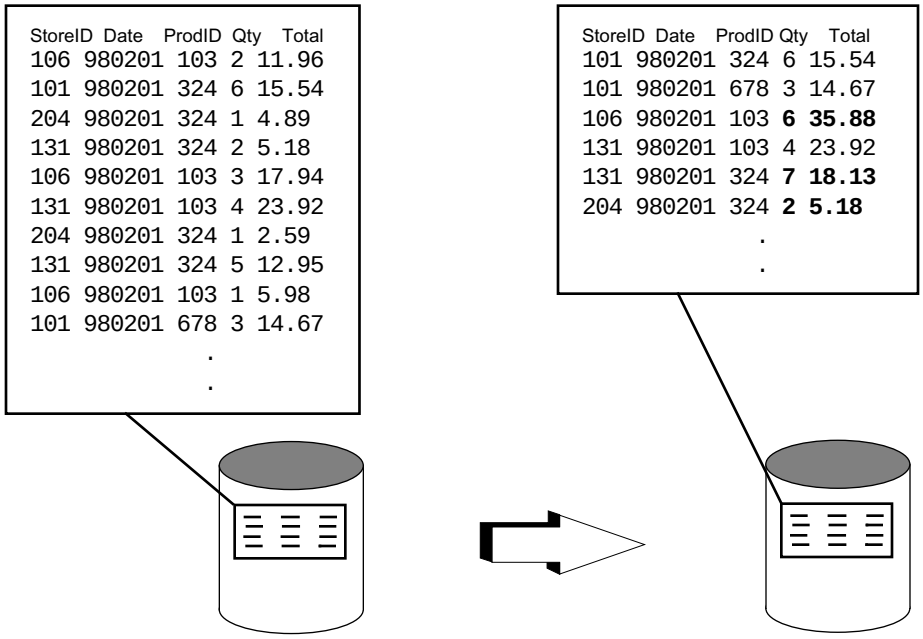
Summarizing Data

The **summarize** command can be used to create summary files containing record fields subtotaled by key value, suitable for loading into summary tables for a data warehouse.

-summarize=*field_name* [**,** *field_name* ...]

A **summarize** statement names fields that will be subtotaled by key value. Records with duplicate keys will be deleted. For each unique key value in the input, there will be one record in the output. Before deleting a record with a duplicate key value, Nsort will add each value in the summarized fields to the corresponding fields in the surviving record.

Figure 5-3 Summarize Data



Summarized fields must be numeric (binary, packed, decimal, float, or double). They can not overlap any key fields.

The following example summarizes quantities and sales for each unique store-date-product combination:

```
-field=StoreID, character,
       Date, character,
       ProdID, character,
       Qty, decimal,
       Sale, decimal
-key=StoreID -key=Date -key=ProdID
-summarize=Qty,Sale
```

The following example assumes fixed-length records, and generates a subtotal of sales for each region:

```
-format=size:8
-field=sales, binary, size:4,
       region, binary, size:4
-key=region
-summarize=sales
```

Before adding the summarized field values to the surviving record, Nsort checks to see if any of the additions will overflow (the result will be too large to fit into the field's size). If so, then Nsort does the following:

- Does not delete the record with duplicate key(s). This may allow both records to appear in the output.
- Does not perform any of the summarize additions for the two records.
- Displays a message warning that a summarize addition overflowed and that duplicate-keyed values may appear in the output (unless the **-no_warnings** option has been used to turn off warnings).

Summarizing Separated Fields

Separated fields that are summarized may need to change sizes in order to hold their summarized values. Nsort automatically increases the temporary, internal size of summarized, separated fields by 10 bytes in order to reduce the occurrence of overflows. This efficiently allows most summarizations to succeed. If this is not appropriate for your application, the **maximum_size**:*number* field option (see "Maximum Field Size" on page 63) can be given. Nsort then temporarily expands those summarized fields to their specified maximum size. For example:

```
# summarize charge amounts by account
-format=separator=comma
-field=account,decimal,
       amount,decimal,maximum_size=22
-key=account
-summarize=amount
```

The above example will expand the "amount" field to be 22 bytes, perform the sort and summarizations, and then trim away unneeded bytes resulting in "amount" fields which are only as large as necessary.

Duplicate Handling

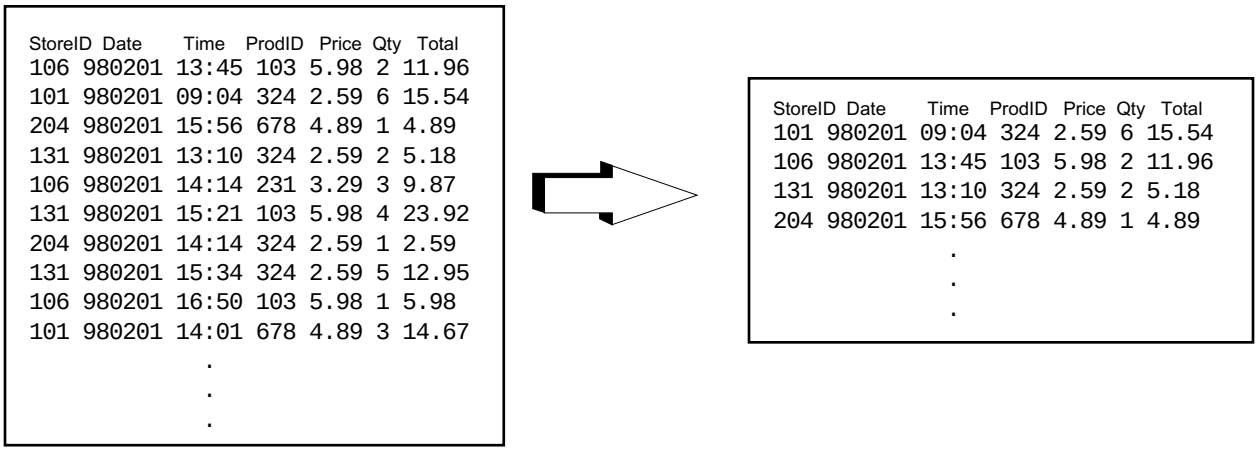
Nsort's default duplicate handling action is to retain all records that have the same key values, returning them in the order they appear in the input (this is sometimes called a ``stable´´ sort). Alternatively, you can direct Nsort to delete all but one record for each set of unique key values.

If **no_duplicates** is specified then the output will not contain multiple records that have the same key fields; all but one of identically-keyed records will be deleted. In a summarizing sort, **duplicates** must not be specified, and **no_duplicates** is ignored.

The following example includes only one record for each store/date combination:

```
-field=StoreID, size:3, character,
       Date, size:6, character,
       ProdID, size:3, character
-key=StoreID -key=Date
-no_duplicates
```

Figure 5-4 Deleting Duplicates



Transforming Data

If you are sorting, merging, or summarizing data, Nsort can transform the data during processing. Nsort can alter record layout, filter data for output files, and derive new fields from existing fields.

Data transformations are described in the following sections:

Selecting Records

You can define include and omit conditions before the first **outfile** statement to specify which input file records are to be included in the sort, and which are to be ignored.

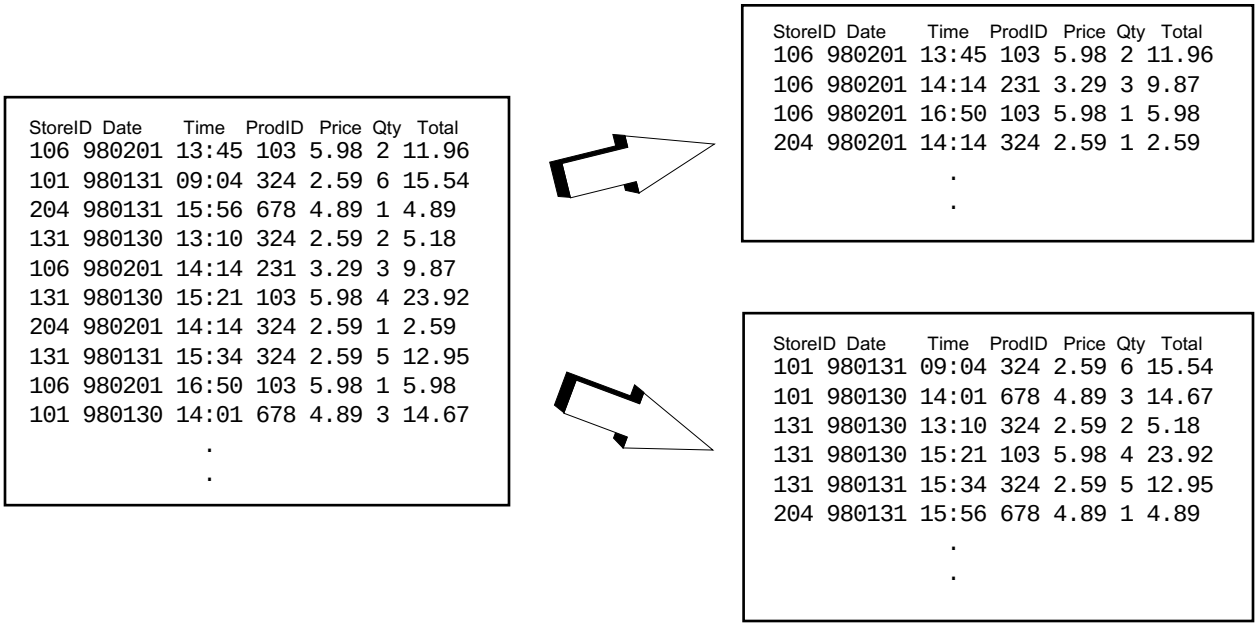
-include=*condition*

-omit=*condition*

The following example includes places records for the past sales in one file and the records for the current day's sales in another file:

```
-field=StoreID, character,
       Date, decimal,
       Time, character
-key=StoreID
-out_file=current_receipts -include=Date == 980201
-out_file=past_receipts -omit=Date == 980201
```

Figure 5-5 Selecting Records



You can specify multiple selection statements. The selection statements are applied, in the order given, until a TRUE condition is found. If the TRUE statement is an **include**, the record is included in the sort. If the TRUE statement is an **omit**, the record is omitted from the sort. Selection statements beyond the TRUE statement are ignored.

If all selection conditions are FALSE, the record is included based on the last selection statement. If the last selection statement was an **omit**, the record is included in the sort. If the last statement was an **include**, the record is omitted from the sort.

The following example includes records with past due balances in excess of 30 days:

```
-field=amount_due, decimal,
       past_due, decimal
-omit=amount_due <= 0 # omit if no amount is due
-omit=past_due < 30    # or if less than 30 days
```

Input file selection cannot currently be done with the **merge** option.

Each **out_file** statement can be followed by selection statements that control the selection of records written to that output file. The series of selection statements for each output file is independent of the selection statements for other output files. That is, the selection of a record in one output file does not affect the selection of the same record in another output file.

The following example places clean records in *clean.dat*, dirty records in *dirty.dat*, and all records in *all.dat*.

```
-cond=clean=(balance >= 0 && balance < 100000000)
-outfile=clean.dat # file for clean records
-include=clean # include clean records for clean.dat
-outfile=dirty.dat # file for dirty records
-omit=clean          # omit clean records for dirty.dat
-outfile=all.dat    # file for all records
```

Count Limitation

For sorts, the **count** statement can be used to limit the number of records read by Nsort.

-**count**:*number*

The count limitation is applied before any record selection.

Reformat

The **reformat** statement defines a new record layout, placing the fields in the specified order.

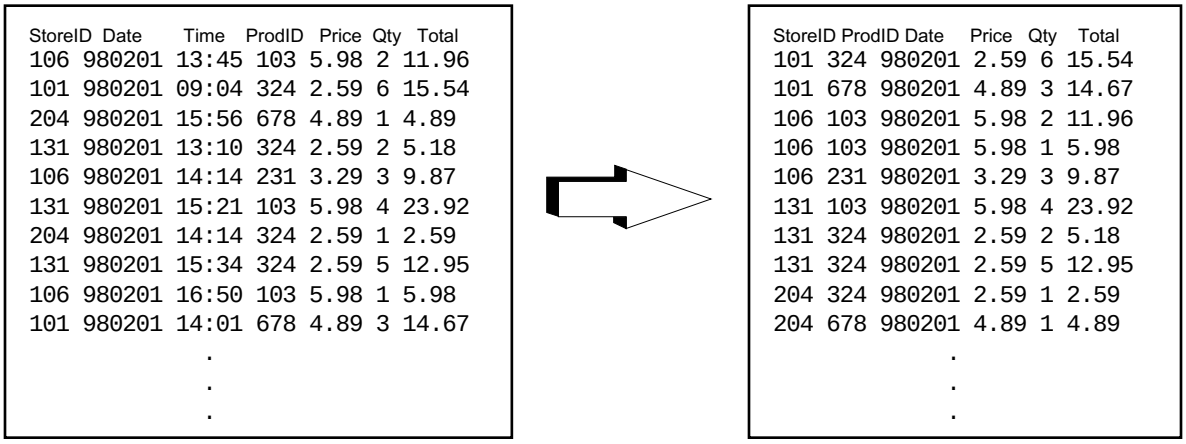
-reformat=*field_name* [**,** *field_name* ...]

The resulting record contains only the listed fields, in the listed order. Nsort can reformat both on input, as it reads records, and on output, as it writes them. Only fixed-length and delimited records can be reformatted.

The following example uses a reformat to remove and reorder fields:

```
-field=StoreID, character,
       Date, character,
       Time, character,
       ProdID, character,
       Price, decimal,
       Qty, decimal,
       Total, decimal
-key=StoreID -key=Date -key=ProdID
-reformat=StoreID, ProdID, Date, Price, Qty, Total
```

Figure 5-6 Reformat Records



If you want to reformat the record for all output files, place the **reformat** statement before the first output file specification. If you want to reformat the records for one particular output file, place the **reformat** statement after the output file statement. If you have multiple output files, the **reformat** statement acts on the output file statement that most recently preceded it.

This section discusses reformatting and includes the following subsections:

- Input reformatting.
- Output reformatting.
- Reformatting guidelines.

Input Reformatting

You can edit records on input by placing the **reformat** statement anywhere before the first output file specification (if any). Nsort performs input reformatting before the sort occurs - key fields and summarized fields must be included in the reformat results. Fields used in input selection statements, however, do not need be present in the reformat field list.

In the following example there are 3 fields in newline-delimited records. The first field is used to select records for the sort. The second field is the sort key. The reformatted records will contain the third and second fields (dropping the first field).

```
-field= a, b, c
-key=b
-omit=(a==”invalid”) # omit record if first field is invalid
-reformat= c, b
```

With input reformatting, keys must be specified using field names (see “Named Keys” on page 65), described keys (“Described Keys” on page 65) are not allowed. If no keys are specified, the reformatted records are ordered as character keys.

Output Reformatting

Each output file can have a distinct record layout. A **reformat** statement after an **out_file** statement defines the field list for the output file.

The following example creates a different layout for each of the sort output files:

```
# lines of text containing whitespace-separated fields
-field=color,                 # first field
       part_number,           # second field
       remainder,position:3-  # remaining input fields
-key=part_number  # order by part_number
# swap color and part_number fields
-reformat=part_number,color,remainder  # input reformat
-condition=is_blue:(color==”blue”)  # true if  color is blue
# put “blue” records in blue output file, dropping color field
-out_file=blue_records
-include=is_blue
-reformat=part_number,remainder  # blue_records file format
# place non-blue records in other file,
# field order is part_number,color,remainder
-outfile=other_records
-omit=is_blue
```

For general information on the **out_file** statement, refer to “Output Files” on page 107.

Changing the Record Format, Field Separator or Record Delimiter

A **format** statement can also be used after an **out_file** statement to specify a new record format, field separator and/or record delimiter for the records written to that output file. A **format** statement may be used with or without a **reformat** statement.

The following example takes lines of text with white space field separators, and changes the separator character for the first output file, and both the separator and record delimiter for the second:

```
-format=delim:nl  # default separator is whitespace
-out_file=out1    # first output file
-format=separator:pipe
-out_file=out2    # second output file
-format=separator:comma,delimiter:null
```

The following example takes fixed-size records containing a binary integer and floating point number, and outputs both fields in ascii as lines of text with the fields separated by a comma character (note that the non-ascii input fields are automatically converted to ascii):

```
-format:size=8
-field:id,binary,size=4
-field:amount,float,size=4
-out_file:out.txt
-format:separator=comma
```

Reformatting Usage Notes

Fixed-Size Records

When sorting lines of text as fixed-length records, a newline field should be included as the last field in a **reformat** statement. The newline field can either be defined in the input record or derived.

The following example reverses the order of two decimal numbers in each record:

```
-format=size:11
-field=a, size:5, decimal,
       b, size:5, decimal,
       nwln, size:1
-key=b
-reformat=b, a, nwln
```

Delimited Records

With delimited records (e.g. lines of text), the following reformat guidelines apply:

- Only separated fields may be used in a **reformat** statement. All fields in a reformatted record will be separated by the field separator.
- An open-end field (a separated field containing a variable number of input fields) can only be used as the last field in a reformat statement.
- Whitespace separated fields will be separated by a single space character after a reformat, even when multiple input fields are contained in a single separated field definition. In addition, the default field value will be substituted for any field that is either empty, contains only whitespace characters, or missing in the input record. Examples of reformats with whitespace-separated fields are given below.

Examples

Given the following record with whitespace-separated fields:

```
red     blue    tan
```

the following field and reformat statements will transform the record as follows:

Directives	Result Record	Comments
`-field=one,two,three,four` `-reformat=one,two,three`	`red blue tan`	single space characters are used to separate reformatted fields
`-field=all,position:1-` `-reformat=all`	`red blue tan`	multiple input fields in a single separated field will be separated by a single space character when reformatted
`-field=one,two,three,four` `-reformat=three,two,one`	`tan blue red`	
`-field=one,two,three,four` `-reformat=four,one`	`*NULL* red`	missing field gets default field value

`-field=one`
`-field=two,position:2.1-2.2`
`-reformat=two,one`
	`*NULL* red`	field containing only whitespace gets default field value

`-field=one`
`-field=two,position:2.1b-2.2`
`-reformat=two,one`
	`*NULL* red`	empty field gets default field value

Derived Fields

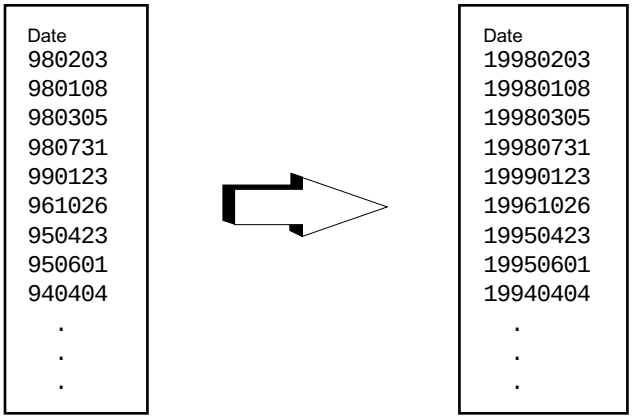
You can use the derived statement to define a new field whose value is either constant or an expression based on other fields. This new field can then be used as a key, in a **summarize** or **reformat** field list, in an expression, or anywhere else that Nsort supports a field name.

-derived=name=*field_name***,** [**size:***number***,**] [*datatype***,**] [**pad:***character***,**]
value:*expression*

A derived field definition is similar to a standard field definition, except for the following:

- The new field's value must be given with a **value=***expression* qualifier.
- Neither **position:***number* nor **offset:***number* are supported.
- The new field cannot be a **bit** type.

Deriving New Fields



Default Addition of Derived Fields

Derived fields are, by default, added to the input record in the order they are declared.

With fixed-size records, the derived fields are appended to the end of the record:

```
# Append a four byte integer to the record
-format=size:4
-field=offset:0, size:4, binary, unsigned
-derived=name:newfield, binary, unsigned, size:4, value:1
```

With delimited records, the derived fields follow all declared fields:

```
# Append new field to the second and first fields
-field=one,
       two
-derived=name:new,char,value="valid"
```

This default behavior can be disabled by specifying the **no_add_derived** statement:

-no_add_derived

With **no_add_derived,** an input **reformat** statement must be used to add derived fields to (or otherwise modify) the records. This statement can be placed in the Nsort system-wide default options file, *nsort.params,* to disable the default appending of derived fields.

A **no_add_derived** statement can be overridden with an **add_derived** statement (the default):

-add_derived

Counting by Key Value

A count of records by key value can be calculated by defining a derived field with a value of 1, then summarizing that field. Two examples of this follow.

In the first example the input is lines of text, each containing a single word. The following statements in a specification file will produce an output file containing the words in sorted order and a count of the number times the work appears in the input:

```
# get count of words in input
-field=word
-derived=name=count,decimal,value="1"
-reformat=word,count
-key=word
-summarize=count
```

In the second example the input is fixed-size records containing a product ID and the amount of an individual sale. The following statements in a specification file will produce an output file containing the product IDs in sorted order, and the total sales and count of sales for that product:

```
-format=size:8
-field=product,binary,size:4
-field=sales,binary,size:4
-derived=name:count,binary,size=4,value=1
-key=product
-summarize=sales,count
```

Expressions

Nsort uses numeric and string expressions to construct derived fields and in the boolean conditions of selection statements.

An Nsort expression can be any of the following:

- A number.
- A string.
- A field name.
- A conditional expression.
- A parenthesized expression.

Numbers in Expressions

Numbers consist of an optional '+' or '-' sign, zero or more digits, an optional decimal point, and zero or more digits. An exponent may also be included.

```
123
-0.1
+3.2E-8
```

Strings in Expressions

Character strings start and end with double quotes ("). You can include non-printable characters by using the escape sequences available for single character constants.

```
"Ted"
"\tHenry\0240Smith"
```

Field Names in Expressions

Field names may be used in nearly any expression. The one exception is that a field which is not included in an input **reformat** may not be used in an expression for an output file. For example:

```
-field:first,second,third
-reformat=second,third      # drop “first” field
-key=second
-include:first != “ignore”  # this is permitted
-out=data.out
-reformat=third
-include=second != “ignore”  # permitted, “first” would not be
```

Conditional Expressions

A conditional expression is an expression of the form:

if *condition* **then** *expression1* **else** *expression2*

The conditional expression evaluates the boolean *condition;* if TRUE it returns *expression1,* otherwise it returns *expression2*. Conditional expressions can be nested and chained together: expressions and conditions can themselves contain conditional components.

```
-derived=name:message, char, size=30,
value:if balance >= 0 then
    if balance == 0 then
        "Thank you for your payment"
    else
        "Credit Balance"
else if past_due < 30 then
    "Please pay this amount"
else
    "Your account is overdue"
```

Conditions

Nsort's boolean conditions appear in conditional expressions and in **include** and **omit** record selection statements. Conditions can be specified and named before they are used, or they can be given inline. A condition is any of the following:

- *condition-name*
- *condition1* { **and** | **&&** } *condition2*
- *condition1* { **or** | **||** } *condition2*
- *expression1 relop expression2*
  where *relop* is one of the following:

  - **EQ** or **==**
  - **NE** or **!=** or **<>**
  - **LE** or **<=**
  - **LT** or **<**
  - **GE** or **>=**
  - **GT** or **>**
  - **CONTAINS** or **CT**
  - **DOES NOT CONTAIN** or **NC**

You can store and name a condition for later use with a condition statement:

-condition:*condition-name* = *expression*

This is especially useful for frequently used and unwieldy conditions like the following:

```
-cond=big_cond:(a == b && c == d) || (a != b && c != d)
-derived=name:new, char, size:10,value:
    if big_cond then
        "matching"
    else
        "different"
```

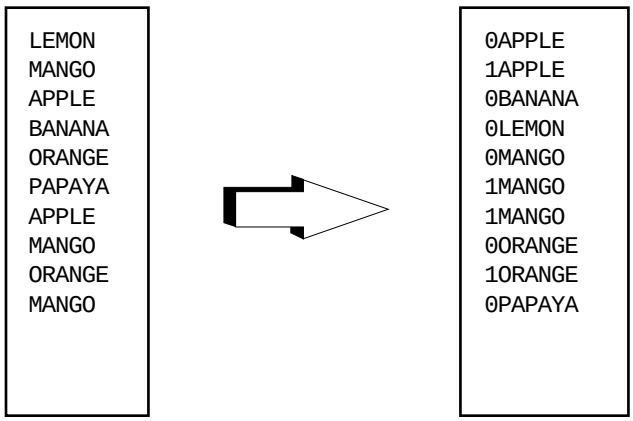
Match Detection

By using the **match** directive, you can have a single character or byte prepended to each output record that indicates whether all or a subset of the keys in the output record match the same keys in the previous output record. If the keys match, the ascii 1 character/byte is prepended in front of the record. An ascii 0 is prepended if the keys do not match.

-match[=*number*]

If a number is specified with the match directive only the specified number of keys, starting with the first key, are compared to generate the prepended character. If no number is specified with the match directive, then all defined keys are compared.

Match Example



Configuration and Performance 6

This chapter covers Nsort's configuration and performance options and includes the following subsections:

Memory Usage

The **memory** option provides many ways to manage Nsort process memory.

-memory=*number*[{**k** | **m** | **g**}]

You can set the limit on the amount of process virtual memory Nsort considers using by specifying the memory size in bytes. Nsort needs memory to hold the sort data plus approximately 10MB for its sorting control structures. Setting the memory limit too high (to more than the amount of available physical memory) can cause excessive page faults or sort termination due to insufficient swap space.

The default memory limit is the minimum of half the amount of system memory currently available for user processes and the following:

- On HP-UX , the resident set size limit for the user running Nsort. The resident set size limit is the value displayed by the ``memoryuse´´ line in the output of the **limit** (csh) command.
- On Linux, Solaris and Windows, there is no additional default memory limit.

If multiple Nsort instances (jobs) are to be run simultaneously in the same system, each Nsort instance should be invoked with a memory statement to limit its use of main memory - otherwise the multiple instances of Nsort can allocate more virtual memory than the available main memory, resulting in page faults and possibly severe performance degradation. For instance to run 4 Nsort jobs simultaneously, each Nsort instance's memory statement should limit it to slightly less than 1/4 of available memory.

Number of Internal Threads

The **threads** option specifies the number of internal sorting threads Nsort will use. For historic reasons, this option can also be specified as **processes** rather than **threads**. The default is to use:

- one sort thread if the input size is known to be less than a megabyte,
- otherwise Nsort creates up to 8 sort threads, but no more than the number of unrestricted processors (see **psradm**(1) on Solaris) in the system.

A warning message is issued if the requested number of threads is larger than the number of processors available to Nsort.

-**threads=***number*

-**processes=***number*

Sort Methods

The **method** statement is used to specify the internal method Nsort shall use to sort the data. It is optional as Nsort chooses a reasonable default.

-method= [{**record** | **pointer**}] [**,** {**merge** | **radix**}] [**, no_radix**] [**, hashing**] [**, no_hashing**]

Record and Pointer Sort

In a record sort, Nsort orders data in memory by moving each record several during in the course of a sort. In contrast, with a pointer sort, Nsort copies a pointer to each record several times, moving the record only once for each sort pass. A record sort can only be performed on fixed-length records of 100 bytes or less.

Record sorts tend to be faster for small records, pointer sorts are faster for large records. By default, Nsort performs a record sort for fixed-length records up to 32 bytes in size (after input record editing). Conversely, larger or non-fixed-length records are sorted using a pointer sort by default.

Merge and Radix Sort

With a merge sort, records or record pointers are brought into sorted order by merging (see *The Art of Computer Programming*, Volume 3, by Knuth, pp. 251-266). Merging is the default. The alternative is a radix sort (Ibid, pp. 170-180), also known as a most significant digit first radix sort, or bucket sort. A radix sort may be somewhat faster than merge when the first 4 bytes of each key have well distributed values. In other cases merging is superior. The radix sort is a deprecated option in Nsort.

Key Hashing

The hashing modifier can be used to speed up some summarizing or duplicate-eliminating sorts, if the records need not be returned in sorted order. The hash modifier changes the semantics of the sort so that records are ordered according to a hash value of the specified keys, rather than the specified keys — but only records with equal key values are treated as duplicates (summarized or deleted). If the first 4 bytes of key values are not well distributed, using the hash modifier may reduce the amount of CPU time necessary to summarize the data or delete duplicates. In other cases it should not be used. The hash modifier can be used with any combination of record or pointer and merge or radix. The hashing modifier is deprecated option in Nsort.

Statistics Output

-[no]statistics

If the **statistics** statement is specified, Nsort will write performance statistics to the standard error file. The default is to not generate statistics.

If you have questions about the performance of a particular sort, you can email the statistics output to support@ordinal.com for analysis. Please include the Nsort command line and any associated Nsort specification files.

Version Information

-version

If the **version** statement is specified, Nsort will write its version, installation directory, processor count and licensing information to the standard error file. Nsort will then terminate.

File I/O

High speed file I/O usually is critical to Nsort performance. High disk transfer rates can be achieved by using file systems comprised of multiple disks. For input and output files, this can be achieved by using a striped logical volume (i.e. a logical disk device that is striped across multiple drives), RAID devices, or a combination of the two.

Nsort allows you to specify several file parameters for I/O performance. The following options are supported wherever Nsort expects a file name:

- The access mode: **direct**, **mapped**, or **buffered**.
- The transfer size: **transfer_size:***number*[**k**|**m**]
- The maximum number of simultaneous I/O requests: **count:***number.*

It is usually not necessary to specify any of these file options as Nsort attempts to select the most appropriate options.

Access Mode

Direct I/O is the most efficient method of accessing files which are not in the UNIX or Windows disk cache. The data is copied directly between the I/O device and Nsort's memory. A **direct** qualifier is most useful for large files which are not primarily resident in the UNIX disk cache.

On Solaris systems, direct i/o is implented via a write-only, persistent file status directive, directio(3S). Nsort will always leave this persistent status as OFF when it finishes execution.

If all or most of an input file is expected to be in the UNIX disk cache, then **mapped** is often the best input mode - the data becomes available to Nsort without any copying at all. **Buffered** access can also be useful for files that are not entirely resident in the UNIX disk cache. Nsort also uses a fourth access mode, **serial**, required for file types for which random access is not possible (e.g. pipes and terminals).

Transfer Size

The transfer size is the amount of data Nsort will request to transfer in any one I/O request to the operating system. A default transfer size is set automatically by Nsort, and may vary depending on the amount of available memory.

Count of Simultaneous I/O Requests

With both **direct** and **buffered** access, Nsort minimizes the amount of time it waits for disk I/O by utilizing asynchronous I/O requests. A default count of overlapping I/O requests is set automatically by Nsort.

File System Defaults

You can specify per-file system defaults for the access mode, transfer size, and request count. Any file residing on such a file system will use the specified value as its defaults.

-file_system= *filename* [**,** {**direct** | **mapped** | **buffered**}]
[**, transfer_size:***number*[**k** | **m**]] [**, count:***number*]

The *filename* can either be the file system name, or the name of any file on the file system (e.g. the mount point).

```
# /xlv is the mount directory for a 64-drive logical volume with
# a step size of 128K. The transfer size is 64*128K or 8M.
-file_sys:/striped,transfer:8M,count:2
```

It is best the let Nsort use its default access mode, transfer size and request count.

Input and Output Files

The **in_file** and **out_file** statements allow input and output files to be declared and, optionally, to set their access mode, transfer size and request count. The specification of an access mode, transfer size or request count on a file overrides any default specified for its corresponding file system.

Input Files

An **in_file** statement names one or more file containing data to be sorted, merged, or copied. A filename consisting of a single dash (-) denotes the standard input. You can give multiple input files by using separate **in_file** statements, or including several file names in one **in_file**, or both. These names are added to any input files listed on the command line to specify the data set to be sorted.

-in_file= *filename*[...] [**,** {**direct** | **mapped** | **buffered**}]
[**, transfer_size:***number*[**k** | **m**]] [**, count:***number*]

It is not necessary to specify the input file access mode, transfer size and count of asynchronous i/o requests. Nsort will make appropriate choices for these parameters.

An example of setting these parameters follows:

```
-in_file=input.data, direct, trans:2M, count:2
```

For file or directory names that contain space characters, the file name should be enclosed in double quotes:

```
-in_file="c:\My Stuff\My Input File.dat"
```

The special wildcard characters '*' and '**?**' can be used to match multiple input files:

```
-in_file=input*.dat
```

Output Files

The output file will be created if necessary. A file specification of a single dash (-) or no **out_file** specification sends the result to the standard output.

-out_file= *filename* [**,** {**buffered** | **direct**}] [**, transfer_size:***number*[**k** | **m**]] [**, count:***number*] [**, append**] [,**preallocate**[:*number*]]

It is not necessary to specify the output file access mode, transfer size and count of asynchronous i/o requests. Nsort will make appropriate choices for these parameters.

The **append** qualifier causes the sorted results to be appended to the end of an existing output file. The default is to replace an existing output file.

The **preallocate** option, available only on Windows platforms, causes Nsort to preallocate file space for the file. The *number* option is a decimal factor which is multiplied by the input file size (or **dataset_size** declaration) to determine the preallocation size. Specifying a factor can be useful in cases where the output file size is different from the input file size (e.g. where duplicates are eliminated). If the preallocation factor is not specified, a factor of 1.0 is used. The **preallocate** option is only useful with very high-bandwidth volumes, and requires SE_MANAGE_VOLUME_NAME privilege.

Example:

```
-out_file=/usr/people/frank/data/sort.output,buffered
```

For file or directory names that contain space characters, the file name should be enclosed in double quotes:

```
-out_file="c:\My Stuff\My Output File.dat"
```

You can assign distinct selection criteria and record layouts for each output file by placing reformat, include, and omit statements after the **out_file** statement. For information on conditionally including/omitting records, reordering fields, or changing the field separator or record delimiter for an output file, refer to "Output Reformatting" on page 86.

Temporary Files

The **temp_file** statement declares the location of the temporary files necessary for a two pass sort. Temporary files are not used if the sort data fits in main memory (one pass sort), or with the **merge** or **cat** options. Temporary files are always accessed using **direct** mode. In the absence of a **temp_file** statement, Nsort creates a single temporary file in the following directory:

- On HP-UX and Linux systems, `/tmp`
- On Solaris systems, `/var/tmp`
- On Windows systems, the default temporary directory returned by the GetTempPath() system call.

The speed of the temporary file systems can be the limiting factor when sorting files residing on striped file systems.

-temp_file= [**default ,**] *filename*[**,**...]
[**, transfer_size:***number*[**k** | **m**]] [**, count**:*number*]
[,**preallocate**[:*number*]]

The **temp_file** statement specifies the stripe set of temporary files for a two pass sort, the transfer sizes to or from these files, and the maximum count of requests to or from each file. If a *filename* refers to a directory then Nsort will create a temporary file in that directory. If a *filename* refers to an existing file, the file will be truncated before it is used. It is recommended the transfer size and count of I/O requests not be specified (and the Nsort defaults used instead).

For file or directory names that contain space characters, the file name should be enclosed in double quotes:

```
-temp="c:\My Temp"
```

The special wildcard characters '*****' and '**?**' can be used to match multiple temporary files:

```
-temp=/temp*
```

The **default** qualifier indicates the list of temporary file directories that follow in the directive will be the default. If a subseqent **-temp_file** directive is specified, the temporary file directories listed in the subsequent directive will replace the default list of directories. In the absence of a **default** qualifier in the first **-temp_file** directive, both lists of temporary file directories will be combined to form the list of temp file directories.

The **preallocate** option, available only on Windows platforms, causes Nsort to preallocate file space for the temporary file. The *number* option is a decimal factor which is multiplied by the input file size (or **dataset_size** declaration) to determine the preallocation size. Specifying a factor can be useful in cases where the temporary file size is different from the input file size (e.g. where duplicates are eliminated). If the preallocation factor is not specified, a factor of 1.0 is used. The **preallocate** option is only useful with very high-bandwidth volumes, and requires SE_MANAGE_VOLUME_NAME privilege.

Multiple Temporary Files

Successive **temp_file** statements accumulate, specifying additional temporary file locations.

```
# declare the directories for Nsort to create its
# temporary file stripe. These happen to be the mount
# directories (but are not required to be) for 4 (presumably
# single-disk) file systems. use a transfer size of 256K,
# and request count of 2.
-temp=/tmp0,/tmp1,/tmp2,/tmp3,transfer:256K,count:2
```

Each temporary file should reside on a separate file system.

Nsort automatically stripes its temporary file data across the collection of temporary files. A run-time error will occur if there is not sufficient space on any one of the temporary file systems. The temporary file space required may be up to 10% more than the size of the input data set (excluding cases where fields are added on input with a reformat statement, or where there is an extremely low amount of main memory available). For example, when sorting a 50 GB input data set with 4 temporary files, the file systems for each temporary file should contain at least one quarter of 55 GB of free space.

It may be useful to include a temp_file statement in the Nsort system-wide defaults file (see "System-wide Default File" on page 21) to declare a system-wide default temporary file stripe. In order to override such a default, a "**-temp_file=( )"** statement will cause all temporary file designations seen up to that point to be discarded, allowing a new set of temporary files to then be defined.

Dataset Size

When one of the files being sorted is a pipe, fifo or tty you can use the **dataset_size** option to easily tune Nsort's memory use.

-dataset_size=*number*[{**k** | **m** | **g**}]

This option specifies the number of bytes that will be in Nsort's input, including any regular files in addition to the pipes, fifos or ttys. Nsort uses the dataset size to determine the appropriate temporary file transfer size and amount of main memory for the sort. Without a dataset size Nsort may use more memory than needed or choose an inappropriately large temporary file transfer size.

This option is unnecessary when all of Nsort's input comes from regular files, or when performing a merge.

Sort Subroutine Library 7

The Nsort subroutine library allows a user application to invoke Nsort subroutines to sort or merge data. The processed data is provided by and returned to the user application. The Nsort subroutine library is available on the HP-UX, Solaris and Windows NT platforms.

This chapter covers the API (application programming interface) for Nsort's sort subroutine library and includes the following subsections:

Compiling and Linking

Compiling and Linking

C programs that invoke Nsort API calls should include the nsort.h file:

```
#include “nsort.h”
```

If the Nsort package has been installed on HP-UX, Linux or Solaris, the nsort.h file has been copied to /usr/include and may be included as:

```
#include <nsort.h>
```

Programs that invoke the Nsort API should be linked with the Nsort library. The following table lists the Nsort library names for each operating system.

HP-UX	libnsort.1
Linux	libnsort.so.1
Solaris	libnsort.so.1
Windows NT	libnsort.lib

Linux and Unix Systems

On Unix systems, the libnsort file is automatically copied to /usr/lib as part of the installation process. An application program can then be compiled and linked as follows:

HP-UX	cc -o *my_app my_app.c* -lnsort -lpthread -lrt -lm
Linux	gcc -o *my_app my_app.c* -lnsort
Solaris	cc -mt -o *my_app my_app.c* -lnsort -lpthread -lposix4 -lm

Windows Systems

When Nsort is installed on Windows the libnsort.dll file is placed in the installation directory, which is added to the default PATH environment variable. The application using the library can be compiled and linked on the command line as follows:

Windows NT cl /MT -o *my_app my_app.c* libnsort.lib

Standard Sort Subroutine Usage

The standard sequence of Nsort function calls to sort records is as follows:

nsort_define(...) to define the sort being performed
for each record or block of records to be sorted:
    nsort_release_recs(...)
nsort_release_end(...) to declare the end of records to be sorted
for each record or block of records to be returned in sorted order:
    nsort_return_recs(...)
nsort_get_stats(...) to get performance statistics (optional)
nsort_end(...) to declare end of sort
nsort_version() to get Nsort library version info (optional)

A simple example program is shown in "Example Application Program Performing a Sort" on page 125.

All of the Nsort API functions return an integer status value that indicates whether the function returned successfully, returned successfully with a warning, or returned with a fatal error. For more details see "Error Handling" on page 138.

The standard Nsort API functions for sorting records will now be explained individually.

nsort_define

```
nsort_msg_t nsort_define(const char *sortdef,
                         unsigned options,
                         nsort_error_callback_t *error_callback,
                         nsort_t *ctxp);
```

The **nsort_define**() call takes a string that describes the sort to be performed and creates a context for the sort that identifies the sort.

Arguments

sortdef	Pointer to a string containing a sort description in Nsort's sort definition language (POSIX sort program command line arguments are not allowed). Input is assumed to come from **nsort_release_recs**() calls by the host program. The host program should use **nsort_return_recs**() to obtain the output records in sorted order.
options	Either 0 (no options) or **NSORT_NO_DEFAULTS** (nsort does not read the nsort.params file to get system-wide configuration defaults.
error_callback	Either NULL (no error callbacks) or points to an nsort_error_callback_t structure that defines an error callback routine and the its first argument. For more details, see "Declaring an Error Callback Routine" on page 140.
ctxp	Pointer to a sort context id (unsigned int). The id is always overwritten with the new sort context id.

Returns

NSORT_SUCCESS	Operation completed successfully.
...	See Appendix A for a complete list.

nsort_release_recs

```
nsort_msg_t nsort_release_recs(void *buf,
                               size_t size,
                               nsort_t *ctxp);
```

The **nsort_release_recs**() call passes one or more records to Nsort for sorting.

Arguments

buf	Pointer to record(s) being released to Nsort for sorting.
size	Integer containing total size in bytes of the records being released
ctxp	Pointer to sort context id.

Returns

NSORT_SUCCESS	Operation completed successfully.
NSORT_INVALID_PHASE	**nsort_release_end**() has already been called for this sort.
NSORT_INVALID_CONTEXT	The context id in invalid.
...	See Appendix A for a complete list.

nsort_release_end

```
nsort_msg_t nsort_release_end(nsort_t *ctxp);
```

The **nsort_release_end**() call indicates that there are no additional records that will be released to Nsort. A **nsort_release_end**() call is required between the last **nsort_release_recs**() call and the first **nsort_return_recs**() call.

Arguments

ctxp	Pointer to sort context id.

Returns

NSORT_SUCCESS	Operation completed successfully.
NSORT_INVALID_CONTEXT	The context id in invalid.
...	See Appendix A for a complete list.

nsort_return_recs

```
nsort_msg_t nsort_return_recs(void *buf,
                              size_t *size,
                              nsort_t *ctxp);
```

The **nsort_return_recs**()call returns one or more output records. Successive calls to **nsort_return_recs**() will return all output records in sorted order.

Arguments

buf	Pointer to buffer where Nsort should place the output records being returned.
size	Pointer to an integer containing the size of the buffer. The size should be at least as large as the maximum declared (or default) record size. The size integer will be modified by the function to contain the total size of the record(s) returned in the buffer.
ctxp	Pointer to sort context id.

Returns

NSORT_SUCCESS	Operation finished successfully.
NSORT_END_OF_OUTPUT	The end of sort output has been reached. No records were returned by this call.
NSORT_INVALID_PHASE	**nsort_release_end**() has not been called for this sort.
NSORT_RETURN_BUF_SMALL	The buffer size is not large enough to hold the declared (or default) maximum record size.
...	See Appendix A for complete list.

nsort_get_stats

```
const char nsort_get_stats(nsort_t *ctxp);
```

The **nsort_get_stats**() call returns a pointer to a character string containing an Nsort statistics report. The user should include the "-statistics" directive in the Nsort command passed to **nsort_define()**. This optional function can only be called after an **nsort_return_recs**() call returns **NSORT_END_OF_OUTPUT**, but before a call to **nsort_end**().

Arguments

ctxp Pointer to sort context id.

Returns

A pointer to a string containing the Nsort statistics report. The memory for the string will be automatically freed when **nsort_end**() is called.

nsort_print_stats

*Note: the **nsort_print_stats**() function has been deprecated in favor of **nsort_get_stats**().*

```
nsort_msg_t nsort_print_stats(nsort_t *ctxp,
                              FILE *fp);
```

The **nsort_print_stats**() call prints the Nsort statistics output to the given file pointer. This optional function can only be called after an **nsort_return_recs**() call returns **NSORT_END_OF_OUTPUT**, but before a call to **nsort_end**().

Arguments

ctxp	Pointer to sort context id.
fp	Standard I/O Library file pointer of file to write the Nsort statistics to.

Returns

NSORT_SUCCESS	Operation completed successfully.
NSORT_INVALID_PHASE	Nsort has not yet returned the final output record.
NSORT_INVALID_CONTEXT	The context id in invalid.
...	See Appendix A for a complete list.

nsort_end

```
nsort_msg_t nsort_end(nsort_t *ctxp);
```

The **nsort_end**() call cancels and/or terminates a sort or merge. This call cancels any sort in progress, frees any resources allocated for the given context, and deallocates the context.

Arguments

ctxp	Pointer to sort or merge context id. The id is cleared by the function before it returns.

Returns

NSORT_SUCCESS	Operation completed successfully.
NSORT_INVALID_CONTEXT	The context id in invalid.
...	See Appendix A for a complete list.

nsort_version

```
char nsort_version(void);
```

The **nsort_version**() call returns a pointer to a character string containing an Nsort version string. This function can be called anytime to identify the current version of the Nsort library.

Arguments

None.

Returns

A pointer to a string containing the Nsort version. The string should *not* be freed by the caller.

Example Application Program Performing a Sort

```
#include <stdio.h>
#include <stdlib.h>
#include "nsort.h"

void error_exit(char *func, int err, unsigned context)
{
    fprintf(stderr, "%s returns %d/%s\n",
            func, err, nsort_message(&context));
    nsort_end(&context);
    exit(1);
}

/* Example program to take input files on the command line,
 * merge the contents and write the result to standard output.
 */
main(int argc, char *argv[])
{
  unsigned  context;
  int       err;
  char      buf[8000];
  int       size;
  unsigned  retsize;

  /* Define sort record type lines of text with fields
   * separated by '|'.  Key is the third field.
   */
  err = nsort_define("-format:sep='|' -key:pos=3",
                     0, NULL, &context);
  if (err < 0)
    error_exit("nsort_define()", err, context);

  /* Read records from standard input, and release them to Nsort
   */
  while (size = read(0, buf, sizeof(buf)))
  {
    if (size < 0)
      perror("<stdin>"), exit(1);
    if ((err = nsort_release_recs(buf, size, &context)) < 0)
      error_exit("nsort_release_recs()", err, context);
  }

  /* Tell Nsort there is no more input. */
  if ((err = nsort_release_end(&context)) < 0)
    error_exit("nsort_release_end()", err, context);
```

```
  /* Get the records output from the sort and write to stdout.
   */
  for (;;)
  {
    retsize = sizeof(buf);  /* Warning: must be bigger than
                               biggest possible record */
    if ((err = nsort_return_recs(buf, &retsize, &context)) < 0)
      error_exit("nsort_return_recs()", err, context);
    if (err == NSORT_END_OF_OUTPUT)
      break;
    write(1, buf, retsize);     /* write to stdout */
  }
  nsort_end(&context);
}
```

Merging Records

The standard sequence of Nsort function calls to merge records is similar to those for sorting records. The main difference is that merge input is provided by a merge input callback routine, rather than **nsort_release_recs**() calls. The sequence of calls for merging is:

nsort_merge_define(...) to define the merge and merge input callback for each record or block of records to be returned in merged order:
    nsort_return_recs(...)
nsort_print_stats(...) to print performance statistics (optional)
nsort_end(...) to declare end of sort

For a simple example program, see "Example Application Program Performing a Merge" on page 131.

The following sections will describe the nsort_merge_define() call, and merge input callback routines.

nsort_merge_define

```
nsort_msg_t nsort_merge_define(const char *sortdef,
                               unsigned options,
                               nsort_error_callback_t *error_callback,
                               int merge_width,
                               nsort_merge_callback_t *merge_input,
                               nsort_t *ctxp);
```

The **nsort_merge_define**() call takes a string that describes the sort to be performed and creates a context for the merge that identifies the merge.

Arguments

sortdef	Pointer to a string containing a merge description in Nsort's sort definition language. Merge input files should not be defined in the string
options	Either 0 (no options) or **NSORT_NO_DEFAULTS** (nsort does not read the nsort.params file to get system-wide configuration defaults.
error_callback	Either NULL (no error callbacks) or points to an nsort_error_callback_t structure that defines an error callback routine and the its first argument.
merge_width	The number of merge input streams to be merged.
merge_input	Pointer to an nsort_merge_callback_t structure that defines a merge input routine and its last argument.
ctxp	Pointer to a sort context id (unsigned int). The id is always overwritten with the new sort context id.

Returns

NSORT_SUCCESS	Operation completed successfully.
...	See Appendix A for a complete list.

Merge Input Callback

A merge input callback routine must be defined in a **nsort_merge_define**() call. Nsort will request merge input records from the application using the merge input callback function.

The merge input callback routine should follow the prototype below:

```
typedef int (*nsort_merge_input_t)(int merge_index,
                                   char *buf;
                                   int size,
                                   void *input_arg);
```

Arguments

merge_index	Index, ranging from 0 to *merge_width* - 1, of the merge input stream being requested.
buf	Pointer to buffer where the requested merge input records should be placed by the callback routine.
size	Size of the *buf* buffer in bytes.
input_arg	Pointer argument specified by the application program in the **nsort_merge_define**() call.

Returns:

> 0	The number of bytes placed in the *buf* buffer.
0	Indicates there is no further input for the given merge stream.
< 0	Indicates an input error has occurred. In this case, Nsort will issue an error message using *errno* on Unix or GetLastError() on Window NT, and then abort the merge.

See the next section for how to declare the merge input function.

Declaring a Merge Input Callback Routine

A merge input callback routine can be declared using the *merge_input_callbacks* argument to **nsort_merge_define**(). This argument is a pointer to the following structure that contains a pointer to the merge input callback routine and its last argument:

```
typedef struct
{
    nsort_merge_input_t     input;
    void                    *arg;
} nsort_merge_callback_t;
```

See previous page for a description of the merge input callback routine.

For more information on **nsort_merge_define**(), see page 128.

Example Application Program Performing a Merge

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include "nsort.h"


/* merge input callback function
 */
int merge_input(int merge_index, char *buf, int size, int *fd)
{
  return (read(fd[merge_index], buf, size));
}


void error_exit(char *func, int err, unsigned context)
{
    fprintf(stderr, "%s returns %d/%s\n",
            func, err, nsort_message(&context));
    nsort_end(&context);
    exit(1);
}




/* Example program to take input files on the command line,
 * merge the contents and write the result to standard output.
 */
main(int argc, char *argv[])
{
  int                    *fd;
  int                    width = argc - 1;
  unsigned               context;
  int                    err;
  nsort_merge_callback_t mc;
  int                    i;
  char                   buf[70000];
  unsigned               retsize;

  fd = malloc(width * sizeof(fd[0]));
  for (i = 0; i < width; i++)
    fd[i] = open(argv[i + 1], O_RDONLY);
```

```
  /* Define merge record type as lines of text with
   * tab-separated field.  Key is second character field.
   */
  mc.input = merge_input;
  mc.arg = fd;
  err = nsort_merge_define("-format:sep=tab -key:pos=2",
                           0, NULL, width, &mc, &context);
  if (err < 0)
    error_exit("nsort_merge_define()", err, context);

  /* Get the records output from the merge and write to stdout.
   */
  for (;;)
  {
    retsize = sizeof(buf);  /* Warning: must be bigger than
                               biggest possible record */
    if ((err = nsort_return_recs(buf, &retsize, &context)) < 0)
      error_exit("nsort_return_recs()", err, context);
    if (err == NSORT_END_OF_OUTPUT)
      break;
    write(1, buf, retsize);     /* write to stdout */
  }
  nsort_end(&context);
}
```

User-Defined Compares

The application program may define its own functions for comparing record fields. Nsort can invoke these comparison functions from multiple threads, thereby leveraging the capabilities of multiprocessor systems.

The following subsections explain the required comparison function type, how to declare a comparison function to Nsort, and how to specify a comparison function in a sort or merge definition.

- Comparison Function Type (page 134).
- Declaring the Comparison Function to Nsort (page 136).
- Specifying the Comparison Function in a Sort or Merge Definition String (page 137).

Comparison Function Type

Comparison functions must be of the following type:

```
typedef int (*nsort_compare_t)(void *p1,
                               void *p2,
                               int len1,
                               int len2,
                               void *compare_arg);
```

Arguments

p1	Pointer the field to be compared in the first record.
p2	Pointer the field to be compared in the second record.
len1	Length in bytes of the field in the first record.
len2	Length in bytes of the field in the second record.
compare_arg	Pointer argument defined when the function was declared using **nsort_declare_function**(). This argument may be used to identify compare derivations of the programmer's choice (e.g. folding upper case to lower case).

Returns

The comparison function should be programmed to return the following values:

< 0	The field in the first record sorts before the field in the second record.
0	The field in the first record sorts equal to the field in the second record.
> 0	The field in the first record sorts after the field in the second record.

Nota Bene

Comparison functions must have the following properties:

- Reentrancy - Nsort will can invoke comparison functions from mutliple threads, even when only one process (thread) is used. Avoid using static variables in comparison functions, or other techniques that will prevent the correct execution of multiple, concurrent comparison functions.
- Trichotomy - For any two key fields **A** and **B**, exactly *one* of the following three relationships must be consistently indicated by the comparison function, regardless the of the order in which the two fields are passed to the comparison function:

  A < B

  A = B

  A > C

  This rule must be followed for all possible key field values, including empty fields (if possible).
- Transitivity - For any three key fields **A**, **B** and **C**, if **A** > **B** and **B** > **C**, then **A** > **C**.

Failure to adhere to these properties can result in Nsort abnormally terminating with an error message, hanging, or even loss of data.

nsort_declare_function

User-defined compare functions can be declared to Nsort using a **nsort_declare_function()** call.

```
nsort_msg_t nsort_declare_function(char *name,
                                   nsort_compare_t function,
                                   void *arg);
```

The **nsort_declare_function**() call is not associated with a particular sort context. The declared function *name* can be used in any subsequent **nsort_define**() or **nsort_merge_define**() call. A previous declaration of the same compare function *name* will be overwritten by **nsort_declare_function**().

Arguments

name	Pointer to a string containing the name of the comparison routine that the application program will use in a subsequent **nsort_define**() call.
function	Pointer to the comparison function.
arg	Compare argument pointer that Nsort will pass to all invocations of *function* to resolve *name* comparisons.

Returns

NSORT_SUCCESS	Operation completed successfully.
...	See Appendix A for a complete list.

Specifying a User-Defined Comparison

The user-defined comparison function declared by an **nsort_declare_function**() call can be used in field comparisons by adding a **compare**:*name* qualifier to the desired field declaration in a subsequent **nsort_define**() or **nsort_merge_define**() call. The *name* in the **compare** qualifier must match the *name* argument to a **nsort_declare_function**() call.

Example

```
int mytype_compare(void *col1, void *col2, int l1, int l2, void *arg);
...
ret = nsort_declare_function("mytype", mytype_compare, mytype_arg);
...
ret = nsort_define("-format=size:72 "
                   "-field:name=fld1,size=4,off=0,compare=mytype "
                   "-key:fldl", 0, NULL, &context);
```

Instead of first declaring a user-defined comparison function with **nsort_declare_function**(), the address of the user-defined comparison function can be given in hexadecimal after the **compare** qualifier. In addition, the value of the *compare_arg* argument to the comparison function can also be declared following the **arg** qualifier.

Example

```
int mytype_compare(void *col1, void *col2, int l1, int l2, void *arg);
struct mystruct myarg;
char def_buf[300];
sprint_s(def_buf, sizeof(def_buf),
        "-format=size:72 -key:off=0,compare=0x%I64x,arg=0x%I64x"
        (unsigned __int64)mytype_compare,
        (unsigned __int64)&myarg);
...
ret = nsort_define(def_buf, 0, NULL, &context);
```

Error Handling

Each Nsort API function returns an integer status value that can indicate success, warning condition or error condition:

< 0	The function did not complete successfully because of a fatal error.
0 (NSORT_SUCCESS)	The function completed successfully.
> 0	The function completed successfully with a non-fatal warning.

Getting an Error or Warning String

When an Nsort function returns non-zero, a warning or error condition has occurred. A character string describing the warning or error can then be obtained by calling **nsort_message**():

```
extern char *nsort_message(nsort_t *ctxp);
```

Arguments

ctxp Pointer to sort context id.

The character string returned should *not* be deallocated by the application program (e.g. by passing the pointer value to **free**()).

Error Callbacks

An error callback routine may be defined to allow application-specific processing of a warning or error condition (e.g. print a non-English description). The callback routine will be called when an Nsort API function detects a warning or error condition, before the API function returns the warning or error status to the application program. The error routine *must* return to allow Nsort to cleanup sort resources.

```
typedef void (*nsort_error_t)(void *arg, nsort_msg_t msg,
                              nsort_t context, char *line
                              int char_no, int n_args, ...);
```

Arguments

arg	Pointer argument declared with callback function.
msg	Integer indicating the type of error or warning that has occurred. This is the same integer that will subsequently be returned by the Nsort API function.
context	Sort context id for which the warning or error is being issued.
line	If the error or warning occurred in parsing the sort specification passed to **nsort_define**() this argument is a pointer to the line where the error occurred, otherwise it is NULL.
char_no	If the error or warning occurred in parsing the sort specification passed to **nsort_define**() this argument is an integer indicating the character number in the line where the error occurred, otherwise it is -1.
n_args	Integer containing the number of additional, error-specific arguments.
...	0 - 7 additional arguments depending on particular error or warning number.

Declaring an Error Callback Routine

An error callback routine can be declared using the *error_callbacks* argument of **nsort_define**(). This argument is either NULL (no error callback routine) or a pointer to the following structure that contains a pointer to the error callback routine and its first argument:

```
typedef struct
{
    nsort_error_t   error;
    void            *arg;
} nsort_error_callback_t;
```

See previous page for a description of the error callback routine.

For more information on **nsort_define**(), see page 117.

Raising an Error in a User-Defined Comparison or Merge Input Callback Function

A user-defined comparison or a merge input callback function can raise a fatal error by calling **nsort_raise_error**().

```
const char nsort_raise_error(char *message);
```

This function will cause the termination of the nsort context associated with the Nsort thread that is calling the compare function or merge input callback.

The function will not return unless the calling thread does not belong to an Nsort context, in which case a descriptive error string is returned.

Errors and Warnings A

This appendix describes the errors and warnings returned by Nsort and includes the following sections:

- Errors.
- Warnings.

The following information is listed for each error or warning message:

- The decimal value returned by Nsort for the error/warning.
- The macro name for the error/warning as given in the nsorterrno.h file.
- The error/warning string printed (or returned by nsort_message()).
- The recommended user action.

Errors

(-1) NSORT_STATEMENT_START
A statement must start with '/' or '-'
Action: Correct the indicated error in the sort specification.

(-2) NSORT_COLON_EXPECTED
A ':' or '=' is expected here
Action: Correct the indicated error in the sort specification.

(-3) NSORT_COMMA_EXPECTED
A ',' is expected inside a list
Action: Correct the indicated error in the sort specification.

(-4) NSORT_UNEXPECTED_IN_LIST
This is not expected inside this list
Action: Correct the indicated error in the sort specification.

(-5) NSORT_PAREN_EXPECTED_BEFORE
A parenthesis '(' was expected here
Action: Correct the indicated error in the sort specification.

(-6) NSORT_PAREN_EXPECTED_AFTER
A parenthesis ')' was expected here
Action: Correct the indicated error in the sort specification.

(-7) NSORT_AMBIGUOUS_IDENTIFIER
This %s name is not sufficiently selective.
The possible names here are:
Action: Correct the indicated error in the sort specification.

(-8) NSORT_UNEXPECTED_KEYWORD
This word is not supported in this context.
The commands recognized here are:
Action: Correct the indicated error in the sort specification.

(-9) NSORT_UNKNOWN_KEYWORD
This is not an Nsort keyword
Action: Correct the indicated error in the sort specification.

(-10) NSORT_IO_ERROR
I/O error occured %s "%s": %s
Action: Analyze OS-issued I/O error and try to correct. For instance, user may not have permission to access/write a file, file system may be out of space or may not support files larger than 2GB.

(-11) NSORT_NEEDS_POSITIVE_INT
A positive integer is expected here
Action: Correct the indicated error in the sort specification.

(-12) NSORT_SCALE_OVERFLOW
This scaled number does not fit in a 64-bit integer
Action: Correct the indicated error in the sort specification.

(-13) NSORT_TYPE_MISMATCH
The type of this expression must match the type of the field
Action: Correct the indicated error in the sort specification.

(-14) NSORT_TYPE_WITHOUT_SIZE
This type needs a size:<number> specification
Action: Correct the indicated error in the sort specification.

(-15) NSORT_EXTENDS_PAST_END
This field extends beyond the end of the record
Action: Correct the indicated error in the sort specification by either increasing the record size of changing the position/offset of the field.

(-16) NSORT_UNSIGNED_WITHOUT_TYPE
The 'unsigned' qualifier is supported only for binary types
Action: Correct the indicated error in the sort specification.

(-17) NSORT_FIELD_NEEDS_EQ
'=' or ':' is needed after 'name'
Action: Correct the indicated error in the sort specification.

(-18) NSORT_METHOD_NEEDS_EQ
'=' or ':' is needed after 'method'
Action: Correct the indicated error in the sort specification.

(-19) NSORT_SPECIFICATION_NEEDS_EQ
'=' or ':' is needed after 'specification'
Action: Correct the indicated error in the sort specification.

(-20) NSORT_FIELD_ALREADY_NAMED
This field has already been named
Action: Correct the indicated error in the sort specification.

(-21) NSORT_BAD_FIELD_NAME
A field name may contains letters, digits, and '_'
Action: Correct the indicated error in the sort specification.

(-22) NSORT_BAD_REC_SIZE
Record sizes may range from 1 to %s
Action: Correct the indicated error in the sort specification.

(-23) NSORT_BAD_REC_SIZE_SPEC
Record size must be followed by an integer or 'variable'
Action: Correct the indicated error in the sort specification.

(-24) NSORT_REC_MUST_BE_VARLEN
This modifier is supported only on non fixed-size records
Action: Correct the indicated error in the sort specification.

(-25) NSORT_MAXLEN_NEEDS_INT
The integral maximum size was expected here
Action: Correct the indicated error in the sort specification.

(-26) NSORT_MAXLEN_INVALID
The maximum size may range from 1 to %s
Action: Correct the indicated error in the sort specification.

(-27) NSORT_MINLEN_NEEDS_INT
The integral minimum size was expected here
Action: Correct the indicated error in the sort specification.

(-28) NSORT_MINLEN_INVALID
The minimum size may range from 1 to %s
Action: Correct the indicated error in the sort specification.

(-29) NSORT_CHARACTER_NEEDED
A character constant (e.g. ',') is expected here
Action: Correct the indicated error in the sort specification.

(-30) NSORT_TYPE_NOT_DELIMITABLE
Delimiters are accepted only for string types
Action: Correct the indicated error in the sort specification.

(-31) NSORT_FIELD_ALREADY_TYPED
This field already has a type
Action: Correct the indicated error in the sort specification.

(-32) NSORT_FLOAT_SIZE
The size of a single precision floating point field is 4 bytes
Action: Correct the indicated error in the sort specification by removing the size specifiler.

(-33) NSORT_DOUBLE_SIZE
The size of a double precision floating point field is 8 bytes
Action: Correct the indicated error in the sort specification by removing the size specifiler.

(-34) NSORT_PACKED_UNIMPLEMENTED
Packed decimal is not yet supported
Action: Correct the indicated error in the sort specification.

(-35) NSORT_FIELD_TOO_SHORT
This field would end before it would start
Action: Correct the indicated error in the sort specification by changing either the field start or end position.

(-36) NSORT_POSITION_NEEDED
An integer value is expected after 'position:'
Action: Correct the indicated error in the sort specification.

(-37) NSORT_DUPLICATE_FIELDNAME
Another field has already been given this name
Action: Correct the indicated error in the sort specification.

(-38) NSORT_FIELD_SIZE_NEEDED
An integer value is expected after 'size:'
Action: Correct the indicated error in the sort specification.

(-39) NSORT_NO_LICENSE
A -license="<license string>" statement is needed to run nsort
Action: Contact Ordinal. As a workaround, get a temporary license from http://www.ordinal.com/temporary.cgi

(-40) NSORT_FIELD_ALREADY_SIZED
The extent of this field has already been specified
Action: Correct the indicated error in the sort specification.

(-41) NSORT_FIELD_ALREADY_DELIMITED
This field already has a delimiter
Action: Correct the indicated error in the sort specification.

(-42) NSORT_FIELD_SYNTAX
Syntax error in field description
Action: Correct the indicated error in the sort specification.

(-43) NSORT_POSITION_POSITIVE
The position of a field starts at 1, not 0
Action: Correct the indicated error in the sort specification.

(-44) NSORT_KEY_FIELD_MISSING
The field named in this key specification is undefined
Action: Correct the indicated error in the sort specification.

(-45) NSORT_KEY_ALREADY_TYPED
This key already has a type defined for it
Action: Correct the indicated error in the sort specification.

(-46) NSORT_MMAP_ZERO_FAILED
Mmap %s bytes of /dev/zero failed: %s; Out of swap space?
Action: Either reduce the memory limit specified to Nsort, or increase the amount of available swap space.

(-47) NSORT_REDUNDANT_ORDERING
Both 'ascending' and 'descending' may not be specified for a single key
Action: Correct the indicated error in the sort specification.

(-48) NSORT_KEY_SYNTAX
Syntax error in key description
Action: Correct the indicated error in the sort specification.

(-49) NSORT_KEY_NUMBER
An integer value is expected after 'number:'
Action: Correct the indicated error in the sort specification.

(-50) NSORT_KEY_NUMBER_INVALID
Key numbers may range from 1 through 255
Action: Correct the indicated error in the sort specification.

(-51) NSORT_KEY_NUMBER_DUPLICATE
This number has already been specified for another key
Action: Correct the indicated error in the sort specification.

(-52) NSORT_DERIVED_NEEDS_VALUE
A derived fields needs a value=<constant> specifier
Action: Correct the indicated error in the sort specification.

(-53) NSORT_FIELD_MISSING
The field "%s" is undefined
Action: Correct the indicated error in the sort specification.

(-54) NSORT_NON_NUMERIC_DERIVED
This derived field cannot have a numeric value
Action: Correct the indicated error in the sort specification.

(-55) NSORT_NON_STRING_DERIVED
This derived field cannot have a string value
Action: Correct the indicated error in the sort specification.

(-56) NSORT_PAGESIZE_POW2
This pagesize is not a power of two >= %s
Action: Correct the indicated pagesize error in the sort specification.

(-57) NSORT_SUMMARIZE_FIELD_MISSING
The summarize field named "%s" was not defined
Action: Correct the indicated error in the sort specification.

(-58) NSORT_OPEN_FAILED
The file "%s" was not opened: %s
Action: Verify that the specified file exists and that the user has access to it.

(-59) NSORT_MEMORY_NEEDED
An integer number of kilobytes of memory to use was expected here
Action: Correct the indicated error in the sort specification.

(-60) NSORT_IOSIZE_UNALIGNED
The i/o transfer size for "%s" must be a multiple of %s bytes
Action: Correct the indicated file transfer size error in the sort specification.

(-61) NSORT_IOSIZE_TOO_LARGE
The maximum i/o transfer size for "%s" is %s bytes
Action: Reduce the indicated file transfer size in the sort specification.

(-62) NSORT_LICENSE_FAILURE
%s: %s
Action: Contact Ordinal. As a workaround, get a temporary license from http://www.ordinal.com/temporary.cgi

(-63) NSORT_LICENSE_MALFORMED
Missing license field; the license format is:
-license="nsort <version> <platform> <serial#> <fmt> [#cpus] <exp. date> <key>"
Action: Contact Ordinal. As a workaround, get a temporary license from http://www.ordinal.com/temporary.cgi

(-64) NSORT_CPU_REQ_NEEDED
An integer number of processors to use was expected here
Action: Correct the indicated error in the sort specification.

(-65) NSORT_NO_VARIABLE_EDITING
Nsort does not yet support editing of variable records
Action: Correct the indicated error in the sort specification.

(-66) NSORT_CAT_ONLY_ONE
Multiple output files are not supported for file copying and concatenation
Action: Correct the indicated error in the sort specification.

(-67) NSORT_FORMAT_SPEC
A record format specification needs either size:<number> or delimiter:<character>
Action: Correct the indicated error in the sort specification.

(-68) NSORT_BAD_METHOD
Known sorting methods are 'record' and 'pointer'
Action: Correct the indicated error in the sort specification.

(-69) NSORT_BAD_HASHSPEC
Known sorting method qualifiers are 'hash' and 'nohash'
Action: Correct the indicated error in the sort specification.

(-70) NSORT_VARIABLE_NEEDS_KEY
A variable size record needs an explicit key definition
Action: Correct the indicated error in the sort specification.

(-71) NSORT_BAD_CHARACTER_SPEC
Unrecognized escape sequence in character constant
Action: Correct the indicated error in the sort specification.

(-72) NSORT_CHARACTER_TOO_LARGE
Oversized escape sequence in character constant
Action: Correct the indicated error in the sort specification.

(-73) NSORT_MISSING_QUOTE
Trailing single quote (') missing
Action: Correct the indicated error in the sort specification.

(-74) NSORT_FILENAME_MISSING
A filename was expected here
Action: Correct the indicated error in the sort specification.

(-75) NSORT_FILESYS_NAME_MISSING
A file or filesystem name was expected here
Action: Correct the indicated error in the sort specification.

(-76) NSORT_SUMMARIZE_DUPLICATES
A summarizing sort may not specify that duplicate keys are to be kept
Action: Correct the indicated error in the sort specification.

(-77) NSORT_NOT_STATEMENT
This is not a statement name
Action: Correct the indicated error in the sort specification.

(-78) NSORT_EXTRA_INSIDE_STATEMENT
A comma was expected here
Action: Correct the indicated error in the sort specification.

(-79) NSORT_EXTRA_AFTER_STATEMENT
Extra characters were found after the end of the statement
Action: Correct the indicated error in the sort specification.

(-80) NSORT_EXTRA_REFORMAT
This is an extra reformat statement
Action: Correct the indicated error in the sort specification.

(-81) NSORT_RECORD_SORTS_VARLEN
Record sorts are supported only for fixed-size records
Action: Correct the indicated error in the sort specification.

(-82) NSORT_RECORD_SORT_TOO_LARGE
Record sorts are limited to no more than %s byte records
Action: Correct the indicated error in the sort specification.

(-83) NSORT_RECORD_TOO_LONG
The record at %s in "%s" is longer than the maximum of %s
Action: Either increase the maximum record size using a -format statement, or make sure that longer records do not appear in the sort input.

(-84) NSORT_RECORD_TOO_SHORT
The record at %s in "%s" is shorter than the minimum of %s
Action: Either decrease the minimum record size using a -format statement, or make sure that shorter records do not appear in the sort input.

(-85) NSORT_REFORMAT_TOO_FAR
Reformatting at the field "%s" would exceed the maximum record size of %s
Action: The reformatting of a particular record in the input would cause the resulting record size to be larger than the maximum record size. Either increase the maximum record size using a -format statement, or make sure that such records do not appear in the sort input.

(-86) NSORT_PARTIAL_RECORD
Data format error: the size of "%s" is not a multiple of the record size
Action: Either fix the specified input file so that it contains a multiple of the (fixed) record size, or change the record size.

(-87) NSORT_DELIM_MISSING
Data format error: the record at %s in "%s" does not have a delimiter
Action: Either first the offending record in the given input file, or change the record delimiter with a -format statement.

(-88) not used

(-89) NSORT_FIELD_BEYOND_END
The field "%s" extends beyond the end of the record
Action: Correct the indicated error in the sort specification.

(-90) NSORT_PAST_MEMORY_LIMIT
There are only %s megabytes of %s available
Action: Either reduce the memory size specified with -memory, or increase the indicated memory limit (per-process memory limit or available swap space).

(-91) NSORT_SPEC_OPEN
Specification file "%s" could not be opened: %s
Action: Check the specification file name, path and permissions.

(-92) NSORT_SPEC_TOO_DEEP
Loop detected: specification file "%s" including "%s"
Action: Verify the named specification file does not reference itself or is part of a cycle of specification files, otherwise reduce the maximum specification file reference depth.

(-93) NSORT_INPUT_OPEN
Input file "%s" could not be opened: %s
Action: Check the input file name, path and permissions.

(-94) NSORT_OUTPUT_OPEN
Output file "%s" could not be opened for writing: %s
Action: Check the output file name, path and permissions.

(-95) NSORT_FILESYS_ERRNO
File or filesystem "%s" could not be accessed: %s
Action: Check the file or filesystem name, path and permissions.

(-96) NSORT_TEMPFILE_STAT
Temp file "%s" could not be opened: %s
Action: Check the temporary file name, path and permissions.

(-97) NSORT_TEMPFILE_BAD_TYPE
Temp file "%s" is not a directory or plain file
Action: Check the temporary file name, path and type.

(-98) NSORT_TEMPFILE_OPEN
Temp file "%s" could not be opened for writing: %s
Action: Verify the named temporary file name and path are correct, and that the user has write permission in the directory.

(-99) not used

(-100) NSORT_FIELD_MAX_ONLY_SEP
Only separated fields may have maximum_size specifications
Action: Correct the indicated error in the sort specification.

(-101) not used

(-102) NSORT_MEMORY_TOO_SMALL
This sort requires at least %sM of memory
Action: Increase the specifiec memory size.

(-103) NSORT_INVALID_CONTEXT
The Nsort context is not valid
Action: Verify the API sort context id is valid.

(-104) NSORT_INVALID_ARGUMENT
A parameter to an Nsort api function is not valid
Action: Look at API call documentation and compare with actual API call arguments to determine which argument is invalid.

(-105) NSORT_FIELD_EXCEEDS_MAX
The field "%s" at %s in "%s" size %s exceeds the maximum size %s
Action: Either increase the field's maximum size or fix the record in the input.

(-106) NSORT_MALLOC_FAIL
Malloc() failed to allocate %s bytes of memory
Action: Verify system is not running low on memory.

(-107) NSORT_APIFILES
An api sort may not specify input or output files
Action: Correct the indicated error in the sort specification.

(-108) not used

(-109) NSORT_KEY_BEYOND_END
The key "%s" extends beyond the end of the record
Action: Correct the indicated error in the sort specification.

(-110) not used

(-111) NSORT_CANCELLED
This sort has been cancelled
Action: None.

(-112) NSORT_CANT_SEEK
The file %s is not seekable and cannot be used for %s i/o
Action: Verify the file name is correct, otherwise change the file access mode to "serial".

(-113) NSORT_INTSIZE
Binary integers may have a size of 1, 2, 4, or 8 bytes
Action: Correct the indicated error in the sort specification.

(-114) NSORT_SUMMARY_NEEDS_NUMBER
The summary field "%s" must have a numeric type
Action: Correct the indicated error in the sort specification.

(-115) NSORT_POSITION_REQUIRED
A position or size qualifier is needed
Action: Correct the indicated error in the sort specification.

(-116) NSORT_SYNTAX_ERROR
Syntax Error
Action: Correct the indicated error in the sort specification.

(-117) NSORT_EXTRA_HEX_DIGIT
Hexadecimal strings must contain an even number of characters
Action: Correct the indicated error in the sort specification.

(-118) NSORT_BAD_HEX_DIGIT
Hexadecimal strings may contain characters only in the range [0-9A-Fa-f]
Action: Correct the indicated error in the sort specification.

(-119) NSORT_STRING_TOO_LONG
This string is too long; the limit is %s bytes
Action: Correct the indicated error in the sort specification.

(-120) NSORT_MISSING_DOUBLE_QUOTE
Trailing double quote (") missing
Action: Correct the indicated error in the sort specification.

(-121) NSORT_TOO_MANY_KEYS
Too many keys were defined: supported limit is at least 4000
Action: Correct the indicated error in the sort specification.

(-122) NSORT_SUMMARIZED_KEY
The summary field "%s" overlaps the key "%s"
Action: Correct the sort specification to not allow an overlap between keys and summarized fields.

(-123) NSORT_MERGE_MISORDERED
Record at %s in file "%s" is misordered
Action: Verify merge specification is correct, otherwise correct out-of-order merge input.

(-124) NSORT_MERGE_NOEDIT
File merging does not yet support input record editing
Action: Correct the indicated error in the sort specification.

(-125) NSORT_APPEND_CHANGED
Output file "%s" changed during sort; append cancelled
Action: Do not allow the output file to be changed by other processes while Nsort is running.

(-126) NSORT_APPEND_NOSTDOUT
Append to standard output is not supported
Action: Correct the indicated error in the sort specification.

(-127) NSORT_MLD_CREATE
The creation of a memory locality domain failed: %s
Action: Contact Ordinal.

(-128) NSORT_MLDSET_CREATE
The creation of the memory locality domain set failed: %s
Action: Contact Ordinal.

(-129) NSORT_MLDSET_PLACE
The placement of the memory locality domain set failed: %s
Action: Contact Ordinal.

(-130) NSORT_TERM_SYNTAX
Syntax error in boolean term
Action: Correct the indicated error in the sort specification.

(-131) NSORT_RE_RANGE_END
Malformed regular expression: range endpoint too large
Action: Correct the indicated error in the sort specification.

(-132) NSORT_RE_NUMBER
Malformed regular expression: bad number
Action: Correct the indicated error in the sort specification.

(-133) NSORT_RE_DIGIT_RANGE
Malformed regular expression: digit out of range
Action: Correct the indicated error in the sort specification.

(-134) NSORT_RE_DELIMITER
Malformed regular expression: illegal or missing delimiter
Action: Correct the indicated error in the sort specification.

(-135) NSORT_RE_REMEMBERED
Malformed regular expression: no remembered search string
Action: Correct the indicated error in the sort specification.

(-136) NSORT_RE_PAREN_IMBALANCE
Malformed regular expression: \\(\\) imbalance
Action: Correct the indicated error in the sort specification.

(-137) NSORT_RE_TOO_MANY_PARENS
Malformed regular expression: too many \\(
Action: Correct the indicated error in the sort specification.

(-138) NSORT_RE_TOO_MANY_NUMS
Malformed regular expression: more than 2 numbers given in \\{ \\}
Action: Correct the indicated error in the sort specification.

(-139) NSORT_RE_BRACE_EXPECTED
**Malformed regular expression: } expected after **
Action: Correct the indicated error in the sort specification.

(-140) NSORT_RE_NUM_PAIR
Malformed regular expression: number exceeds second in \\{ \\}
Action: Correct the indicated error in the sort specification.

(-141) NSORT_RE_BKT_IMBALANCE
Malformed regular expression: [ ] imbalance
Action: Correct the indicated error in the sort specification.

(-142) NSORT_RE_EXPBUF_OVERFLOW
Malformed regular expression: regular expression overflow
Action: Correct the indicated error in the sort specification.

(-143) NSORT_GENERATE_COUNT
A record count=N is needed when generating records
Action: Correct the indicated error in the sort specification.

(-144) NSORT_REFORMAT_FIELD_MISSING
The field "%s" is not available for reformatting
Action: Verify the field name is correct and has been defined.

(-145) NSORT_EXPECTED_THEN
The keyword "then" is expected after "if" <condtion>
Action: Correct the indicated error in the sort specification.

(-146) NSORT_FIELD_REMOVED
The field "%s" was removed by a prior /reformat
Action: Correct the indicated error in the sort specification.

(-147) NSORT_EXPECTED_ELSE
The keyword "else" is expected after "if" ... "then"
Action: Correct the indicated error in the sort specification.

(-148) NSORT_TYPE_CHANGED
Reformat changed types incompatibly: %s to %s
Action: Correct the indicated error in the sort specification.

(-149) NSORT_REFORMAT_KEY_MISSING
The key "%s" must be included by input reformatting
Action: Correct the indicated error in the sort specification.

(-150) NSORT_EXCESSIVE_CONSTANT
The type "%s" is not large enough to contain %s
Action: Correct the indicated error in the sort specification.

(-151) NSORT_RECURSIVE_DERIVED
The value of "%s" may not refer to itself
Action: Correct the indicated error in the sort specification.

(-152) NSORT_REFORMAT_COND
The condition "%s" may not be named in a reformat list
Action: Correct the indicated error in the sort specification.

(-153) NSORT_MONTH_TOO_SHORT
A field of type "month" must be at least 3 characters long
Action: Correct the indicated error in the sort specification.

(-154) NSORT_FILTER_INCOMPAT_KEYS
A filter or copy may not have any keys
Action: Correct the indicated error in the sort specification.

(-155) NSORT_INCOMPAT_RECORD_TYPE
Separated fields are only supported in separated records
Action: Correct the indicated error in the sort specification.

(-156) NSORT_PADTYPES_DIFFER
Two strings have different pad chracters %s, %s in "%s"
Action: Correct the indicated error in the sort specification.

(-157) NSORT_RECORD_MTBUF_SIZE
The record at %s in "%s" is too large for the calculated maximum_size of %s
Action: Either 1) explicitly increase the transfer size of the temp file (for a sort) or input files (for a merge) to be larger than twice the calculated maximum record size given in the error message, 2) increase the Nsort memory limit, or 3) reduce the maximum size of the records in the input.

(-158) NSORT_UNSUPPORTED_MERGE_SELECTOR
Merge does not support input selection
Action: Correct the indicated error in the sort specification.

(-159) NSORT_MISSPELLED_KEYWORD
Unexpected token; perhaps a command name has been misspelled?
Action: Check the spelling of keywords in the sort specification.

(-160) NSORT_STRING_EXPECTED
A quoted string is expected here
Action: Correct the indicated error in the sort specification.

(-161) NSORT_FILTER_CANNOT_GROW
Record copying currently does not allow records to become larger
Action: Correct the indicated error in the sort specification.

(-162) NSORT_UNUSED_REFORMAT
Record copying ignores input reformatting when output reformatting is also specified
Action: Correct the indicated error in the sort specification.

(-163) NSORT_BAD_DELIM_MOD
Unrecognized field modifier; valid ones are "bdfiMnr "
Action: Limit field modifiers to the character list in the error message.

(-164) NSORT_COL_DOT_OFF
A delimited field is specified as <column_number>[.<character_offset]
Action: Either drop the field-specific delimiter, or define the field as starting at a fixed position from the beginning of the record.

(-165) NSORT_IMPLICIT_DERIVED_REFORMAT
Derived fields such as "%s" must be explicitly added to non fixed-size records
Action: Create a reformat statement that includes the dervied field and other desired fields.

(-166) NSORT_BAD_FIELD_SIZE
Field sizes may range from 1 to 65535 bytes
Action: Reduce the field size to 65,535 (or 8MB for the Windows versions of Nsort).

(-167) NSORT_REFORMAT_KEY_UNNAMED
A key ("%s") must refer to a named field when reformatting the input record
Action: Correct the indicated error in the sort specification.

(-168) NSORT_REFORMAT_FIXED_STREAM
Stream records do not support reformatting of fixed sized fields such as "%s"
Action: Correct the indicated error in the sort specification.

(-169) NSORT_DELIMFIELDS_DEPRECATED
Delimited fields are no longer supported
Action: Correct the indicated error in the sort specification.

(-170) NSORT_ALREADY_POSITIONED
This field already has been positioned
Action: Use only one position/offset for the field.

(-171) NSORT_REFORMAT_DELIMITED
The delimited field "%s" may not be reformatted
Action: Correct the indicated error in the sort specification.

(-172) NSORT_KEY_EXCEEDS_MAX
The key "%s" at %s in "%s" size %s exceeds the maximum size %s
Action: Either increase the key's maximum size or fix the record in the input.

(-173) NSORT_REMAINDER_REFORMAT
The field "%s" contains an unknown number of subfields and may be placed only at the end of the reformat list
Action: Correct the indicated error in the sort specification.

(-174) NSORT_BINARY_IN_DELIMITED
Delimited records cannot contain fields of type %s
Action: Correct the indicated error in the sort specification.

(-175) NSORT_SUMMARIZE_NEEDS_KEY
No key was given for a summarizing sort
Action: Correct the indicated error in the sort specification.

(-176) NSORT_LOGICAL_EXPR_NEEDED
The selector "%s" needs a logical expression
Action: Correct the indicated error in the sort specification.

(-177) NSORT_UNSUPP_CHANGE_RECORD
Nsort does not support changing between fixed-size and stream records
Action: Correct the indicated error in the sort specification.

(-178) NSORT_CANNOT_DETERMINE_POSITION
Nsort needs an position specification for this field
Action: Correct the indicated error in the sort specification.

(-179) NSORT_OUT_OF_SWAP
Insufficient space to map %sMB of memory: %s
Action: Either reduce the Nsort memory limit or free up system swap space.

(-180) NSORT_RESOURCE_LIMITED
Resource limits constrained Nsort to only %dMB of memory
Action: Either reduce the Nsort memory limit or increase the process memory resource limit (using the shell).

(-181) NSORT_32BIT_LIMITED
This 32-bit Nsort is constrained to use at most %sMB of memory
Action: Either reduce the Nsort memory limit, or use the 64-bit version of Nsort.

(-182) NSORT_SYSCALL
API system call failed %s %s
Action: Contact Ordinal.

(-183) NSORT_INVALID_PHASE
API call out of sequence; expected %s != %s
Action: Make sure API calls are done in normal order.

(-184) NSORT_LOCK_FAILED
API call: mutex_lock failed; %s
Action: Contact Ordinal.

(-185) not used

(-186) NSORT_UNLOCK_FAILED
API call: mutex_unlock failed; %s
Action: Contact Ordinal.

(-187) NSORT_COMPARE_UNDECLARED
The comparison function "%s" has not been declared
Action: Correct the indicated error in the sort specification.

(-188) NSORT_FOLD_POINTER
The 'fold_upper' and 'fold_lower' modifiers are currently supported only for pointer sorts
Action: Don't specify a sort method of "record" to use fold_upper or fold_lower.

(-189) NSORT_THREAD_CREATE
Nsort could not create a thread: %s
Action: Contact Ordinal.

(-190) NSORT_PWRITE64_ERROR
pwrite64 system call does not work on this Solaris 2.7 server, please install Solaris patch 106980-18
Action: Install specified Solaris patch.

(-191) NSORT_BAD_MERGE_INPUT
The merge input callback function is not defined
Action: Define a merge input callback function in the nsort_merge_define() call.

(-192) NSORT_BAD_MERGE_WIDTH
The merge width has not been specified
Action: Define a merge width in the nsort_merge_define() call.

(-193) NSORT_API_DEFINE_MERGE
Merge is not allowed with nsort_define(), use nsort_merge_define() instead
Action: Drop the -merge statement with nsort_define(), or use nsort_merge_define() to initiate a merge.

(-194) NSORT_API_MERGE_INPUT
Input file definitions are not allowed with nsort_merge_define()
Action: Remove input file specifications with nsort_merge_define().

(-195) NSORT_UNKNOWN_SCALE
Invalid numeric scale in "%s"; supported values are [kmg]
Action: Modify the scale suffix for the number being specified.

(-196) NSORT_NUMBER_OVERFLOW
The number in "%s" is too large for a 64-bit integer
Action: Modify the specified number.

(-197) NSORT_NEEDS_STRING
A character string is needed here
Action: Provide the needed character string.

(-198) NSORT_EXTRA_DOES
Syntax error: this operator already has a 'does' modifier
Action: Delete the redundant does.

(-199) NSORT_EXTRA_NOT
Syntax error: this operator already has 'not' modifier
Action: Delete the redundant not.

(-200) NSORT_NOT_UNSUPPORTED
Syntax error: this operator does not support the 'not' modifier
Action: Delete the unsupported not.

(-201) NSORT_IN_REC_SIZE_REQUIRED
The input format statement needs an integer record size
Action: Provide the record size in the format statement.

(-202) NSORT_BAD_BIT_FIELD
Illegal bit field declaration
Action: Correct the bit field declaration.

(-203) NSORT_BIT_POSITION_TOO_LARGE
Bit field offsets can range from 0 through 7
Action: Correct the bit field offset to 0-7 or position to 1-8.

(-204) NSORT_BIT_SIZE_TOO_LARGE
Bit field sizes cannot exceed their position:
Action: Reduce the bit field size to fit be within a byte.

(-205) NSORT_REFORMAT_NEEDS_MAXSIZE
Please specify a maximum size for the variable string field "%s"
Action: Provide the maximum size for the named field.

(-206) NSORT_VALUE_CONTAINS_SPECIAL
The "%s" value "%s" may not contain the %s
Action: Correct the nsort value.

(-207) NSORT_PARTIAL_VARIABLE_RECORD
The last record "%s" (offset %s) would extend beyond the end of the file
Action: Fix record at the end of the file.

(-208) NSORT_EXTRA_FORMAT
The format of this file has already been specified
Action: Remove the duplicate format statement.

(-209) NSORT_SIZE_BEYOND_LICENSE
This sort is larger than the license key allows
Action: Reduce the input size or obtain a license with a larger input size limit.

(-210) NSORT_PACKED_SIZE_TOO_LARGE
The size of a packed decimal field cannot exceed 31
Action: Correct the size of the packed decimal field.

(-211) NSORT_PACKED_OVERFLOW
The number in "%s" is too large for a packed decimal field
Action: Correct to the too large number.

(-212) NSORT_MERGE_INPUT_CALLBACK_OVERWRITE
The merge input callback function for input stream %s offset %s size %s wrote beyond the end of the buffer
Action: Correct the merge input callback function to not overwrite the buffer.

(-213) NSORT_USER_RAISED_ERROR
User-defined comparison or merge input callback error: %s
Action: Fix the error raised by the comparison or callback function.

Warnings

(1) NSORT_END_OF_OUTPUT
All records have been returned
Action: None.

(2) NSORT_CLOSE_FAILED
The file "%s" was not closed: %s
Action: None.

(3) NSORT_UNLINK_FAILED
The temp file "%s" could not be removed: %s
Action: None.

(4) NSORT_MEMORY_MINIMAL
This operation appears to need %sMB of memory; excessive paging is possible; continuing
Action: Increase Nsort's memory limit for better performance.

(5) NSORT_EXCESSIVE_PAGING
Performance caution: Excessive paging (%s faults) detected
Action: Nsort's memory limit (specified or default) is larger than the amount of physical memory available to it. Try reducing the memory limit.

(6) NSORT_REDUCING_IOSIZE
The default i/o transfer size of %s is too large for available memory; reducing to %s
Action: Either eliminate input file transfer size specifications for the merge, or reduce them to the result size indicated in the warning message.

(7) NSORT_PMTRACE_PROBLEM
Peformance Co-Pilot pmdatrace error for %s: %s
Action: Contact Ordinal.

(8) NSORT_SUMMARIZE_OVERFLOWED
Some summarizations would have overflowed; the output may contain duplicate keys
Action: Increase the maximum field size of the summarized field so that all unique sets of keys will have only one record in the output.

(9) not used

(10) NSORT_CPU_REQ_TOOBIG
This system's processor count is only %s
Action: Reduce the requested processor count to the number indicated in the warning message.

(11) NSORT_CPU_REQ_RESTRICTED
This system has only %s out of %s processors available to you
Action: Reduce the requested processor count to the number indicated in the warning message.

(12) NSORT_RECSORT_OUTFILES
Multiple output files requires pointer sorts; continuing
Action: Either delete the -method=record directive, or specify only one output file.

(13) NSORT_POINTER_SORT_ONLY
Output file editing and selection require pointer sorts; continuing
Action: Either delete the -method=record directive, or delete the include, omit and reformat directives.

(14) NSORT_TEMPFS_INAPPROPRIATE
%s is on a tmpfs filesystem; this may cause poor performance
Action: Don't specify a paging-based temporary file system on Solaris (e.g. /tmp) for the the temporary file. For better performance (especially with large data sets) use normal file systems such as /var/tmp.

(15) NSORT_CONVERSIONS_OVERFLOWED
The values of %s expressions were too large to fit in their derived fields
Action: Increase the size of a derived field to avoid overflow.

(16) not used

(17) not used

(18) NSORT_MLDLINK
Process_mldlink() failed in sproc %s: %s
Action: Contact Ordinal.

(19) NSORT_RUNANYWHERE
Sysmp(RUNANYWHERE) failed in sproc %s: %s
Action: Contact Ordinal.

(20) NSORT_UNDEFINED_KEY
The semantics for the key %s are not defined
Action: Using the Unix sort sematics for specifying the ending position of a key, do not specify both a last chararter of 0 and the skip blanks modifier ('b').

(21) NSORT_KEY_TOO_SHORT
The key "%s" starts after it ends
Action: Correct the indicated error in the sort specification.

(22) NSORT_DETAIL_IO
Error writing detailed statistics log "%s" at %s bytes: %s; continuing without detail log
Action: Contact Ordinal.

(23) NSORT_RETURN_BUF_SMALL
The return buf is too small (size %s) to hold the next result record (size %d)
Action: Either increase the size of the buffer passed to nsort_return_recs() so that it is large enough to hold the maximum record size, or reduce the maximum record size using -format=maximum:N.

(24) NSORT_IGNORING_KEY
The key "%s" is the same as "%s"; ignoring it
Action: Delete the duplicate use of a key in the sort specification.

(25) NSORT_DELIMITER_ADDED
The file "%s" did not end with the record delimiter [but %s]; one has been added
Action: Make sure the input file is correct and the correct record delimiter is specified. If necessary, add a record delimiter to the end of the input file.

(26) NSORT_PREALLOCATE
Preallocation failure "%s" : %s
Action: Make sure you have the correct privilege for preallocation (SE_MANAGE_VOLUME_NAME for Windows systems).

Index

D

E

F

P

Q

R

V

W