Compaq C Run-Time Library Utilities Reference Manual

Order Number: AA–R238C–TE

April 2001

This manual documents Compaq C Run-Time Library utilities used in developing international software applications that manage localization and time zone data.

Revision/Update Information: This manual supersedes the *Compaq C Run-Time Library Utilities Reference Manual*, Version 7.1.

Software Version: OpenVMS Alpha 7.3
OpenVMS VAX Version 7.3

Compaq Computer Corporation
Houston, Texas

ZK6494

The Compaq *OpenVMS* documentation set is available on CD-ROM.

This document was prepared using DECdocument, Version 3.3-1b.

Contents

3 Character Set Description (Charmap) File

4 Command Reference

Index

Tables

Preface

The *Compaq C Run-Time Library Utilities Reference Manual* provides detailed usage and reference information about Compaq C Run-Time Library utilities for managing localization and time zone data in international software applications.

Intended Audience

This manual is for programmers who use the Compaq C Run-Time Library to develop applications that manage localization and time zone data.

Document Structure

This manual consists of four chapters.

- Chapter 1 introduces the GENGAT, ICONV, LOCALE, and ZIC programming utilities.
- Chapter 2 explains the locale definition source file and describes the standard locale categories that Compaq supports.
- Chapter 3 discusses the character set description (`charmap`) file, which defines character symbols as character encodings.
- Chapter 4 provides complete command descriptions for the XPG4 and ZIC utilities.

Related Documents

The following documents provide additional information about the Compaq C Run-Time Library utilities:

- *Compaq C Run-Time Library Reference Manual for OpenVMS Systems*
- *OpenVMS Version 7.0 New Features Manual*

For additional information about Compaq *OpenVMS* products and services, access the Compaq website at the following location:

`http://www.openvms.compaq.com/`

Reader's Comments

Compaq welcomes your comments on this manual. Please send comments to either of the following addresses:

Internet	**openvmsdoc@compaq.com**
Mail	Compaq Computer Corporation OSSG Documentation Group, ZKO3-4/U08 110 Spit Brook Rd. Nashua, NH 03062-2698

How to Order Additional Documentation

Use the following World Wide Web address to order additional documentation:

`http://www.openvms.compaq.com/`

If you need help deciding which documentation best meets your needs, call 800-282-6672.

Conventions

The following conventions are used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
Return	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.) In the HTML version of this document, this convention appears as brackets, rather than a box.
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.
{ }	In command format descriptions, braces indicate required choices; you must choose one of the options listed. Do not type the braces in the command line.
bold text	This typeface represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.

italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
Monospace text	Monospace type indicates code examples and interactive screen displays. In the C programming language, monospace type in text identifies the following elements: keywords, the names of independently compiled external functions and files, syntax summaries, and references to variables or identifiers introduced in an example.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

1
Overview

The Compaq C Run-Time Library utilities help you to manage localization and time zone data for international software applications. Localization and time zone data is defined separately from the application and is bound to it only at run time.

The Compaq C Run-Time Library includes the following utilities:

- XPG4-compliant utilities (Section 1.1)
- ZIC utility (Section 1.2)

1.1 Creating XPG4-Compliant Localizing Applications

To help you develop localizing applications for use internationally, the OpenVMS operating system offers, as part of its Compaq C Run-Time Library, several utilities that support the XPG4 model (X/Open Portability Guide Issue 4) of internationalization. The following XPG4-compliant utilities are provided:

- GENCAT utility (Section 1.1.1)
- ICONV utility (Section 1.1.2)
- LOCALE utility (Section 1.1.3)

These tools are useful only for applications written to the XPG4 model.

1.1.1 Creating and Invoking Message Catalogs

A message catalog is a binary file that contains the messages an application displays or writes. This file includes all the messages that the application issues, for example, error messages, information messages, screen displays, and prompts. To create message catalogs, use the GENCAT command.

GENCAT reads one or more input source files and the existing catalog file, if one exists. The source file is a text file that you create to hold the messages that your program might print. Use any text editor to enter messages into the source file. If you identify multiple source files, GENCAT processes them one after the other in the sequence that you specify them. Each successive source file modifies the catalog.

Before you or your application issues GENCAT, create the required input source file and, if appropriate at this time, the catalog file.

For more detailed information about the GENCAT command, see Chapter 4.

1.1.1.1 Message Source File

When you create an input source file, follow these guidelines:

- Group your messages into sets to represent functional subsets of the program.
- Give each message a numeric identifier, which must be unique within its set.
- Add commands recognized by GENCAT for manipulating sets and individual messages.

1.1.1.2 Message Catalog File

If a message catalog with the name *catfile* exists, GENCAT creates a new version of the file that includes the contents of the older version and then modifies it. If the catalog does not exist, GENCAT creates it with the name *catfile*.

1.1.1.3 Retrieving Messages from a Message Catalog

OpenVMS applications retrieve messages from a message catalog using the following Compaq C Run-Time Library routines:

- `catopen`
- `catgets`
- `catclose`

For details, see the *Compaq C Run-Time Library Reference Manual for OpenVMS Systems*.

1.1.2 Performing Codeset Conversions

The ICONV utility provides the following commands to create a conversion table file from a conversion source file and, using this file, to convert characters from one codeset to another:

- The ICONV COMPILE command creates a conversion table file.
- Using this conversion table file, the ICONV CONVERT command then converts characters in another, specified file from one codeset to another.

The ICONV commands support any 1- to 4-byte codesets that are state independent.

Note

There is a restriction in the *tocodeset* encodings in this implementation. The characters in *tocodeset* must not use 0XFF in the fourth byte.

1.1.2.1 Creating Conversion Tables

To create a conversion table file, issue the DCL command ICONV COMPILE:

ICONV COMPILE *sourcefile tablefile*

See the description of the ICONV COMPILE command in Chapter 4 for the format of the conversion source file.

See the description of the ICONV CONVERT command in Chapter 4 for the tablefile naming convention.

1.1.2.2 Converting from One Codeset to Another

To convert characters in a file from one codeset to another codeset, issue the ICONV CONVERT command:

ICONV CONVERT *infile outfile* /FROMCODE=*fromcodeset* /TOCODE=*tocodeset*

The converted characters are written to the output file parameter *outfile*.

1.1.3 Setting International Environment Logical Names

The LOCALE utility is an OpenVMS XPG4 localization utility that:

- Compiles a binary locale file for use by utilities and C routines dependent on the setting of the international environment logical names
- Loads a locale name into system memory as shared, read-only global data
- Displays a summary of the current international environment as defined on your system and details of locales on your system
- Unloads a locale name from system memory

The LOCALE utility supports the following commands:

- The LOCALE COMPILE command converts a locale source file into a binary locale file for use by utilities and C routines. This command allows you to add new locales to your system in addition to those specified by Compaq.

  To compile a locale, the LOCALE COMPILE command uses two source files:

  - A locale definition source file that contains categories that describe a locale. Locale categories, described in Table 4–3, include LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.
  - A character set description (charmap) file that defines the character set for the locale. The charmap, which defines character symbols as character encodings, is the source file for a coded character set (see Chapter 3).
- The LOCALE LOAD command loads a locale name into system memory as several shared, read-only, global sections. All processes that access the loaded locale use this one copy of the locale, thereby reducing overall demand on system memory.
- The LOCALE UNLOAD command unloads a specified locale name from system memory.
- The LOCALE SHOW CHARACTER_DEFINITIONS command lists the names of the character set description files (charmaps) in the public directory defined by the logical name SYS$I18N_LOCALE. A charmap defines the symbolic names and values of characters in a coded character set. A charmap file has the file type .CMAP.
- The LOCALE SHOW CURRENT command displays a summary of the current international environment as defined by several logical names representing locale categories. This command lists the settings for each locale category and the values of the environment variables LC_ALL and LANG. The logical name that defines a category has the same name as the category. For example, the LC_MESSAGES logical name defines the setting for the LC_MESSAGES category.

- The LOCALE SHOW PUBLIC command lists all the public locales on the system, including locales listed in the directory defined by the logical name SYS$118N_LOCALE as well as system locales supplied by the Compaq C Run-Time Library.
- The LOCALE SHOW VALUE command displays the value of one or more keywords from the current international environment. Locale categories and keywords in each category are listed in Table 4–4.

For more information about LOCALE commands, see Chapter 4.

1.2 Creating Time Zone Conversion Information

Using the Zone Information Compiler (ZIC) utility, the ZIC compiler creates binary files containing time zone conversion information. These files are generated from the time zone source files that you specify.

The lines in the source files consist of fields. To create a valid time zone source file, follow these formatting requirements:

- Any number of white space characters separate the fields.
- Leading and trailing white spaces on input lines are ignored.
- An unquoted number sign (**#**), the sharp character, in the input line introduces a comment that extends to the end of the line where this sign appears.
- White space characters and sharp characters can be enclosed in double quotation marks (" ") if they are to be used as part of a field.
- Any line that is blank after comment stripping is ignored.
- Non-blank lines are expected to be one of three types:
  - Rule lines (see Section 1.2.1)
  - Zone lines (see Section 1.2.2)
  - Link lines (see Section 1.2.3)

1.2.1 Rule Lines

A rule line has the following form:

```
Rule  NAME  FROM    TO TYPE IN ON      AT   SAVE LETTER/S
```

An example is as follows:

```
Rule  USA   1969 1973 - Apr lastSun    2:00 1:00 D
```

The rule line consists of the following fields:

NAME

Gives the arbitrary name of the set of rules that this rule is part of.

FROM

Gives the first year in which the rule applies. The word minimum, or an abbreviation, means the minimum year with a representable time value. The word maximum, or an abbreviation, means the maximum year with a representable time value.

TO

Gives the final year in which the rule applies. In addition to minimum and maximum as defined in FROM, minimum or maximum (or an abbreviation) only may be used to repeat the value of the FROM field.

TYPE

Gives the type of year in which the rule applies. If TYPE is –, then the rule applies in all years between FROM and TO inclusively. ZIC executes the following command to check the type of year:

```
yearistype year type
```

An exit status of 1 means that the year is of the given type; an exit status of 5 means that the year is not of the given type.

IN

Gives the month in which the rule takes effect. Month names may be abbreviated.

ON

Gives the day on which the rule takes effect. Table 1–1 shows the recognized forms.

Table 1–1 Day the Rule Becomes Effective

Form	Meaning
5	Fifth of the month
lastSun	Last Sunday in the month
lastMon	Last Monday in the month
Sun>=8	First Sunday on or after the 8th
Sun<=25	Last Sunday on or before the 25th

Names of days of the week may be abbreviated or spelled out in full. Note that there must be no spaces within the ON field.

AT

Gives the time of day when the rule takes effect. Table 1–2 shows the recognized forms.

Table 1–2 Time of Day the Rule Becomes Effective

Form	Meaning
2	Time in hours
2:00	Time in hours and minutes
15:00	24-hour format time (for times after noon)
1:28:14	Time in hours, minutes, and seconds

Any of these forms may be followed by the letter **w** if the given time is local *wall clock* time, or the letter **s** if the time is local *standard* time. In the absence of either the letter **w** or the letter **s**, *wall clock* time is assumed.

SAVE

Gives the amount of time to be added to local standard time when the rule is in effect. This field has the same format as the AT field, although, of course, the letter **w** and **s** suffixes are not used.

LETTER/S

Gives the *variable part* of time zone abbreviations to be used when this rule is in effect; as for example, the *S* or *D* in *EST* or *EDT*. If this field is –, the variable part is null.

1.2.2 Zone Lines

A zone line has the following form:

```
Zone NAME              GMTOFF  RULES/SAVE FORMAT UNTIL
```

An example is as follows:

```
Zone Australia/South-west 9:30   Aus         CST  1987 Mar 15 2:00
```

The zone line consists of the following fields:

NAME

Gives the name of the time zone. This name is used in creating the time conversion information file for the zone.

GMTOFF

Gives the amount of time to add to Greenwich mean time (GMT) to get standard time in this zone. This field has the same format as the AT and SAVE fields of rule lines. If time must be subtracted from GMT, begin the field with a minus sign.

RULES/SAVE

Gives the name of the rule(s) that apply in the time zone, or alternatively, an amount of time to add to local standard time. If this field is –, standard time always applies in the time zone.

FORMAT

Gives the format for time zone abbreviations in this time zone. The pair of characters **%s** is used to show where the variable part of the time zone abbreviation goes.

UNTIL

Gives the time at which the GMT offset, or the rule(s) change for a location. It is specified as the following:

A year
A month
A day
A time of day

If UNTIL is specified, the time zone information is generated from the given GMT offset and rule change until the time specified.

If you specify UNTIL, the next line must be a *continuation* line. The continuation line has the same form as the zone line except that the string *Zone* and the name are omitted, for the continuation line places information starting at the time specified in the UNTIL field in the previous line in the file used by the previous line. Continuation lines may contain an UNTIL field, just as zone lines do, indicating that the next line is a further continuation.

1.2.3 Link Lines

A link line has the following form:

```
Link      LINK-FROM      LINK-TO
```

An example is as follows:

```
Link      US/Eastern      EST5EDT
```

In the OpenVMS implementation, Link is interpreted as a copy. Thus, the previous line copies the information from US/Eastern to EST5EDT.

The LINK-FROM field should appear as the NAME field in some zone line. The LINK-TO field is used as an alternate name for that zone.

Except for continuation lines, lines may appear in any order in the input.

Note

For areas with more than two types of local time, use local standard time in the AT field of the earliest transition time's rule to ensure that the earliest transition time recorded in the compiled file is correct.

2
Locale File Format

A locale definition source file contains categories that describe a locale. You can convert a locale definition source file into a locale by using the LOCALE COMPILE command. Locales can be modified only by editing a locale definition source file and then using the LOCALE COMPILE command again on the new source file. Each locale source file section defines a category of locale data. A source file cannot contain more than one section for the same category.

2.1 Locale Categories

The following standard locale categories are supported:

- LC_COLLATE — Defines character or string collation information
- LC_CTYPE — Defines character classification, case conversion, and other character attributes
- LC_MESSAGES — Defines the format for affirmative and negative responses
- LC_MONETARY — Defines rules and symbols for formatting monetary numeric information
- LC_NUMERIC — Defines rules and symbols for formatting nonmonetary numeric information
- LC_TIME — Defines rules and symbols for formatting time and date information

2.1.1 Overriding Defaults

You can include optional declarations at the beginning of your locale source file to override the default comment and escape characters used in locale category definitions:

- Escape character

  The escape character is used in decimal or hexadecimal constants when they are specified in the locale file. The default escape character is the backslash (\). To define another escape character, include a line with the following format:

  ```
  escape_char  <char_symbol>
  ```

- Comment character

  The comment character is the first character of each comment entry in the locale file. The default comment character is the number sign (#). To define another comment character, use the following format:

  ```
  comment_char  <char_symbol>
  ```

In the preceding formats, <*char_symbol*> is the character's symbolic name as defined in the charmap file used to build the locale's codeset. One or more blank characters (spaces or tabs) must separate escape_char or comment_char from <*char_symbol*>.

2.1.2 Category Source Definitions

Each category source definition consists of the following:

- The category header (*category_name*)
- The associated keyword or value pairs that comprise the category body
- The category trailer (END *category_name*)

For example:

```
LC_CTYPE
<source for LC_CTYPE category>
END LC_CTYPE
```

The source for all of the categories is specified using keywords, strings, character literals, and character symbols. Each keyword identifies either a definition or a rule. The remainder of the statement containing the keyword contains the operands to the keyword. Operands are separated from the keyword by one or more blank characters (spaces or tabs). A statement may be continued on the next line by placing a backslash (\) as the last character before the new-line character that terminates the line. Lines containing the comment character (#) in the first column are treated as comment lines.

A symbolic name begins with the left angle-bracket character (<) and ends with the right angle-bracket character (>). The characters between the < and the > can be any characters from the Portable Character Set, except for the control and space characters. For example, <A-diaeresis> could be a symbolic name for a character. Any symbolic name referenced in the locale source file must be defined via the Portable Character Set or in the character set description (charmap) file for that locale.

A character literal is the character itself, or a decimal, hexadecimal, or octal constant. A decimal constant contains two or three decimal digits and has the following form, where *n* is any decimal digit:

\d*nn* or \d*nnn*

A hexadecimal constant contains two hexadecimal digits and has the following form, where *n* is any hexadecimal digit:

\x*nn*

An octal constant contains two or three octal digits and has the following form, where *n* is any octal digit:

nn or *nnn*

The explicit definition of each category in a locale definition source file is not required. When a category is undefined in a locale definition source file, the LOCALE COMPILE command will not store any data value for this category in the resulting locale file.

2.2 LC_COLLATE Category

The LC_COLLATE category defines the relative order between collation items. This category begins with the LC_COLLATE header and ends with the END LC_COLLATE trailer.

A collation item is the unit of comparison for collation. A collation item may be a character or a sequence of characters. Every collation item in the locale has a set of weights, which determine if the collation item collates before, equal to, or after the other collation items in the locale. Each collation item is assigned collation weights by the LOCALE COMPILE command when the locale definition source file is compiled. These collation weights are then used by applications programs that compare strings.

String comparison is performed by comparing the collation weights of each character in the string until either a difference is found or the strings are determined to be equal. This comparison may be performed several times if the locale defines multiple collation orders. For example, in the French locale, the strings are compared using a primary set of collation weights. If they are equal on the basis of this comparison, they are compared again using a secondary set of collation weights. A collation item has a set of collation weights associated with it that is equal to the number of collation sort rules defined for the locale.

Every character defined in the charmap file (or every character in the Portable Character Set if no charmap file is specified) is itself a collation item. Additional collation items can be defined using the `collating-element` statement (see the description that follows).

Table 2–1 lists the statement keywords recognized in the LC_COLLATE category.

Table 2–1 LC_COLLATE Category Keywords

Keyword	Description
`copy`	Specifies the name of an existing locale to be used as the definition of this category. If you specify a copy statement, you need not specify any other keywords in this category.
`collating-element`	Specifies multicharacter collation items.
`collating-symbol`	Specifies collation symbols for use in collation sequence statements.
`order_start`	Specifies collation order statements that assign collation weights to collation items.

The `collating-element`, `collating-symbol`, and `order_start` statements are further described in the following sections.

2.2.1 The collating-element Statement

The `collating-element` statement specifies multicharacter collation items.

Syntax:

```
collating-element <character_symbol>
from <string>
```

The *character_symbol* argument defines a collation item that is a string of one or more characters as a single collation item. The *character_symbol* cannot duplicate any symbolic name in the current charmap file or any other symbolic name defined in this collation definition.

The *string* argument specifies a string of two or more characters that define the *character_symbol* argument. The following are examples of the syntax for the `collating-element` statement:

```
collating-element <ch> from "<c><h>"
collating-element <e-acute> from "<acute><e>"
collating-element <11> from "<1><1>"
```

A *character_symbol* argument defined by the `collating-element` statement is recognized only within the LC_COLLATE category.

2.2.2 The collating-symbol Statement

The `collating-symbol` statement specifies collation symbols for use in collation sequence statements.

Syntax:

`collating-symbol <`*`collating_symbol`*`>`

The `collating-symbol` argument cannot duplicate any symbolic name in the current charmap file or any other symbolic name defined in this collation definition. The following are examples of `collating-symbol` statements:

```
collating-symbol <UPPER_CASE>
collating-symbol <HIGH>
```

An argument defined by the `collating-symbol` statement is recognized only within the LC_COLLATE category.

2.2.3 The order_start Statement

The `order_start` statement is followed by one or more collation order statements that assign collation weights to collation items and the `order_end` keyword. The `order_start` statement is a required statement.

Syntax:

```
order_start sort_rules;sort_rules;...;sort_rules
collation_order_statements
order_end
```

Sort Rules

The *sort_rules* directives have the following syntax:

```
keyword, keyword,...,keyword
```

where *keyword* is FORWARD, BACKWARD, or POSITION.

The *sort_rules* directives are optional. If specified, they define the rules to apply during string comparison. The number of specified *sort_rules* directives defines the number of weights each collation item is assigned (that is, the directives define the number of collation orders in the locale). If no *sort_rules* directives are specified, one forward directive is assumed and comparisons are made on a character basis rather than a string basis.

If *sort_rules* directives are present, the first one applies when comparing strings that use the primary weight, the second when comparing strings that use the secondary weight, and so on. Each set of *sort_rules* directives is separated by a semicolon (;). A *sort_rules* directive consists of one or more keywords separated by commas. The following keywords are supported:

FORWARD — Specifies that collation weight comparisons proceed from the beginning of a string to the end of the string.

BACKWARD — Specifies that collation weight comparisons proceed from the end of a string to the beginning of the string.

POSITION — Specifies that collation weight comparisons consider the relative position of nonignored elements in the string (that is, if strings compare as equal, the element with the shortest distance from the starting point of the comparison collates first).

The forward and backward keywords are mutually exclusive.

The following is an example of a *sort_rules* directive:

```
order_start         forward;backward
```

Collation Order Statements

The following syntax rules apply to the collation order statements:

- Each collation order statement consists of a <*character_symbol*> specification followed by white space and a set of collation orders.
- Characters in the character set can be explicitly specified in the collation order statements or implicitly specified using the ellipsis symbol (...).
- A collation order statement that begins with the UNDEFINED special symbol specifies any characters that are in the character set but not explicitly or implicitly specified by other collation order statements.

The optional operands for each collation item are used to define the primary, secondary, or subsequent weights for the collation item. The special symbol IGNORE is used to indicate a collation item that is to be ignored when strings are compared.

An ellipsis keyword appearing in place of a collating_element_list indicates the weights are to be assigned, for the characters in the identified range, in numerically increasing order from the weight for the character symbol on the left side of the preceding statement.

The use of the ellipsis keyword results in a locale that may collate differently when compiled with different character set description (charmap) source files.

The UNDEFINED special symbol includes all coded character set values not specified explicitly or with an ellipsis symbol. These characters are inserted in the character collation order at the point indicated by the UNDEFINED special symbol and are all assigned the same weight. If no UNDEFINED special symbol exists and the collation order does not specify all collation items from the coded character set, a warning is issued and all undefined characters are placed at the end of the character collation order.

Example

The following is an example of a collation order statement section in the LC_COLLATE locale definition source file category:

```
order_start     forward;backward
UNDEFINED       IGNORE;IGNORE
<LOW>
<space>         <LOW>;<space>
...              <LOW>;...
<a>             <a>;<a>
<a-acute>       <a>;<a-acute>
<a-grave>       <a>;<a-grave>
<A>             <a>;<A>
<A-acute>       <a>;<A-acute>
<A-grave>       <a>;<A-grave>
<ch>            <ch>;<ch>
<Ch>            <ch>;<Ch>
<s>             <s>;<s>
<ss>            <s><s>;<s><s>
<eszet>         <s><s>;<eszet><eszet>
...             <HIGH>;...
<HIGH>
order_end
```

This example is interpreted as follows:

- The UNDEFINED special symbol indicates that all characters not specified in the definition (either explicitly or by the ellipsis symbol) are ignored for collation purposes.
- All collation items between <space> and <a> have the same primary equivalence class and individual secondary weights based on their coded character-set values.
- All versions of the letter a (uppercase and lowercase, and with or without diacriticals) belong to the same primary collation class.
- The <c><h> multicharacter collation item is represented by the <ch> collating symbol and belongs to the same primary equivalence class as the <C><h> multicharacter collation item.
- The <eszet> character is collated as an <s><s> string (that is, one <eszet> character is expanded to two characters before comparing).

2.3 LC_CTYPE Category

The LC_CTYPE category defines character classification, case conversion, and other character attributes. This category begins with the LC_CTYPE header and ends with the END LC_CTYPE trailer.

All operands for LC_CTYPE category statements are defined as lists of characters. Each list consists of one or more characters or symbolic character names separated by semicolons. An ellipsis (...) can represent a series of characters; for example, <a>;...;<z> represents the characters in the range a through z.

Table 2–2 lists the statement keywords recognized in the LC_CTYPE category. In the keyword descriptions, the phrase “automatically included” means that an error does not occur if the referenced characters are included or omitted; the characters are provided if they are missing, and are accepted if they are present.

Table 2–2 LC_CTYPE Category Keywords

Keyword	Description
copy	Specifies the name of an existing locale to be used as the definition for this category. If you specify a copy statement, you cannot specify any other keyword.
upper	Defines uppercase letter characters. Do not specify any character defined by the cntrl, digit, punct, or space keyword. The uppercase letters A through Z are automatically included in this set.
lower	Defines lowercase letter characters. Do not specify any character defined by the cntrl, digit, punct, or space keyword. The lowercase letters a through z are automatically included in this set.
alpha	Defines all letter characters. Do not specify any character defined by the cntrl, digit, punct, or space keyword. Characters defined by the upper and lower keywords are automatically included in this character class.
digit	Defines numeric digit characters. Only the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 can be specified. The digits 0 through 9 are automatically included in this set.
space	Defines white-space characters. Do not specify any character defined by the upper, lower, alpha, digit, graph, or xdigit keyword. The space, form-feed, new-line, carriage-return, tab, and vertical tab characters are automatically included in this set.
cntrl	Defines control characters. Do not specify any character defined by the upper, lower, alpha, digit, punct, graph, print, or xdigit keyword.
punct	Defines punctuation characters. Do not specify the space character or any character defined by the upper, lower, alpha, digit, cntrl, or xdigit keywords.
graph	Defines printable characters, excluding the space character. Do not specify any character defined by the cntrl keyword. The characters defined by the upper, lower, alpha, digit, xdigit, and punct keywords are automatically included in this character class.
print	Defines printable characters, including the space character. Do not specify any character defined by the cntrl keyword. The space character and characters defined by the upper, lower, alpha, digit, xdigit, and punct keywords are automatically included in this character class.
xdigit	Defines hexadecimal digit characters. Only the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 can be specified. Any character, however, can be specified for the hexadecimal values for 10 to 15. These alternate hexadecimal digits are not used by standard conversion routines when converting digit strings from hexadecimal to numeric quantities. The numbers 0 through 9 and the letters A through F and a through f are automatically included in this set.

(continued on next page)

Table 2–2 (Cont.) LC_CTYPE Category Keywords

Keyword	Description
blank	Defines blank characters. The space and horizontal tab characters are included in this character class. Any characters defined by this statement are automatically included in the space class.
toupper	Defines the mapping of lowercase characters to uppercase characters. Operands for this keyword consist of character pairs separated by commas. Each character pair is enclosed in parentheses () and separated from the next pair by a semicolon (;). The first character in each pair is considered a lowercase character; the second character is considered an uppercase character. Only characters defined by the lower and upper keywords can be specified. If toupper is not specified, a through z is mapped to A through Z by default.
tolower	Defines the mapping of uppercase characters to lowercase characters. Operands for this keyword consist of character pairs separated by commas. Each character pair is enclosed in parentheses () and separated from the next pair by a semicolon (;). The first character in each pair is considered an uppercase character; the second character is considered a lowercase character. Only characters defined by the lower and upper keywords can be specified. If tolower is not specified, the mapping defaults to the reverse mapping of the toupper keyword, if specified. If the toupper and tolower keywords are both omitted, the mapping for each defaults to that of the C locale.

Additional keywords can be provided to define new character classifications. For example:

```
charclass vowel
vowel       <a>;<e>;<i>;<o>;<u>;<y>
```

The LC_CTYPE category does not support multicharacter elements (for example, the German Eszet character is traditionally classified as a lowercase letter). In proper capitalization of German text, the Eszet character is replaced by the two characters SS; there is no corresponding uppercase letter. This kind of conversion is outside the scope of the toupper and tolower keywords.

The following is a sample LC_CTYPE category specified in a locale definition source file:

```
LC_CTYPE
#"alpha" is by default "upper" and "lower"
#"alnum" is by definition "alpha" and "digit"
#"print" is by default "alnum", "punct" and the space character
#"graph" is by default "alnum" and "punct"
#"tolower" is by default the reverse mapping of "toupper"
#
upper   <A>;<B>;<C>;<D>;<E>;<F>;<G>;<H>;<I>;<J>;<K>;<L>;<M>;\
        <N>;<O>;<P>;<Q>;<R>;<S>;<T>;<U>;<V>;<W>;<X>;<Y>;<Z>
#
lower   <a>;<b>;<c>;<d>;<e>;<f>;<g>;<h>;<i>;<j>;<k>;<l>;<m>;\
        <n>;<o>;<P>;<q>;<r>;<s>;<t>;<u>;<v>;<w>;<X>;<y>;<z>
#
digit   <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
        <seven>;<eight>;<nine>
#
space   <tab>;<newline>;<vertical-tab>;<form-feed>;\
        <carriage-return>;<space>
#
cntrl   <alert>;<backspace>;<tab>;<newline>;<vertical-tab>;\
        <form-feed>;<carriage-return>;<NUL>;<SOH>;<STX>;\
        <ETX>;<EOT>;<ENQ>;<ACK>;<SO>;<SI>;<DLE>;<DC1>;<DC2>;\
        <DC3>;<DC4>;<NAK>;<SYN>;<ETB>;<CAN>;<EM>;<SUB>;\
        <ESC>;<IS4>;<IS3>;<IS2>;<IS1>;<DEL>
#
punct   <exclamation-mark>;<quotation-mark>;<number-sign>;\
        <dollar-sign>;<percent-sign>;<ampersand>;<asterisk>;\
        <apostrophe>;<left-parenthesis>;<right-parenthesis>;\
        <plus-sign>;<comma>;<hyphen>;<period>;<slash>;\
        <colon>;<semicolon>;<less-than-sign>;<equals-sign>;\
        <greater-than-sign>;<question-mark>;<commercial-at>;\
        <left-square-bracket>;<backslash>;<circumflex>;\
        <right-square-bracket>;<underline>;<grave-accent>;\
        <left-curly-bracket>;<vertical-line>;<tilde>;\
        <right-curly-bracket>
#
xdigit  <zero>;<one>;<two>;<three>;<four>;<five>;<six>;\
        <seven>;<eight>;<nine>;<A>;<B>;<C>;<D>;<E>;<F>;\
        <a>;<b>;<c>;<d>;<e>;<f>
#
blank   <space>;<tab>
#
toupper (<a>,<A>);(<b>,<B>);(<c>,<C>);(<d>,<D>);(<e>,<E>);\
        (<f>,<F>);(<g>,<G>);(<h>,<H>);(<i>,<I>);(<j>,<J>);\
        (<k>,<K>);(<l>,<L>);(<m>,<M>);(<n>,<N>);(<o>,<O>);\
        (<P>,<P>);(<q>,<Q>);(<r>,<R>);(<s>,<S>);(<t>,<T>);\
        (<u>,<U>);(<v>,<V>);(<w>,<W>);(<X>,<X>);(<y>,<Y>);\
        (<z>,<Z>)
#
END LC_CTYPE
```

2.4 LC_MESSAGES Category

The LC_MESSAGES category defines the format for affirmative and negative system responses. This category begins with the LC_MESSAGES header and ends with the END LC_MESSAGES trailer.

All operands for the LC_MESSAGES category are defined as strings or extended regular expressions bounded by double quotation marks ("). These operands are separated from the keyword they define by one or more blank characters (spaces or tabs). Two adjacent double quotation marks ("") indicate an undefined value.

Table 2–3 lists the statement keywords recognized in the LC_MESSAGES category.

Table 2–3 LC_MESSAGES Category Keywords

Keyword	Description
copy	Specifies the name of an existing locale to be used as the definition of this category. If you specify a copy statement, you cannot specify any other keyword.
yesexpr	Specifies an extended regular expression that describes the acceptable affirmative response to a question expecting an affirmative or negative response.
noexpr	Specifies an extended regular expression that describes the acceptable negative response to a question expecting an affirmative or negative response.
yesstr	Specifies the locale's equivalent of an acceptable affirmative response. This string is accessible to applications through the nl_langinfo subroutine as nl_langinfo (YESSTR). Note that yesstr is likely to be withdrawn from the XPG4 standard; yesexpr is the recommended alternative.
nostr	Specifies the locale's equivalent of an acceptable negative response. This string is accessible to applications through the nl_langinfo subroutine as nl_langinfo (NOSTR). Note that nostr is likely to be withdrawn from the XPG4 standard; noexpr is the recommended alternative.

The following is a sample LC_MESSAGES category specified in a locale definition source file:

```
LC_MESSAGES
#
yesexpr "<circumflex><left-square-bracket><y><Y>\
<right-square-bracket>"
noexpr  "<circumflex><left-square-bracket><n><N>\
<right-square-bracket>"
yesstr  "<y><e><s>"
nostr   "<n><o>"
#
END LC_MESSAGES
```

2.5 LC_MONETARY Category

The LC_MONETARY category defines rules and symbols for formatting monetary numeric information. This category begins with the LC_MONETARY header and ends with the END LC_MONETARY trailer.

2.5.1 LC_MONETARY Keywords

All operands for the LC_MONETARY category keywords are defined as string or integer values. String values are bounded by double quotation marks ("). All values are separated from the keyword they define by one or more blank characters (spaces or tabs). Two adjacent double quotation marks ("") indicate an undefined string value. A negative one (-1) indicates an undefined integer value.

Table 2–4 lists the statement keywords recognized in the LC_MONETARY category.

Table 2–4 LC_MONETARY Category Keywords

Keyword	Description
copy	Specifies the name of an existing locale to be used as the definition of this category. If you specify a copy statement, you cannot specify any other keyword.
int_curr_symbol	Specifies the string used for the international currency symbol. The operand for this keyword is a 4-character string†. The first three characters contain the alphabetic international currency symbol. The fourth character defines a character separator for insertion between the international currency symbol and a monetary quantity.
currency_symbol	Specifies the string used for the local currency symbol.
mon_decimal_point	Specifies the decimal delimiter string used for formatting monetary quantities.
mon_thousands_sep	Specifies the character separator used for grouping digits to the left of the decimal delimiter in formatted monetary quantities.
mon_grouping	Specifies a string that defines the size of each group of digits in formatted monetary quantities. The operand for this keyword consists of a sequence of integers separated by semicolons. Each integer specifies the number of digits in a group. The first integer defines the size of the group immediately to the left of the decimal delimiter. Subsequent integers define succeeding groups to the left of the previous group. If the last integer is not -1, it is used to group any remaining digits. If the last integer is -1, no further grouping is performed. A sample interpretation of the mon_grouping statement follows. Assuming a value of 123456789 to be formatted and a mon_thousands_sep operand of ' (single quotation mark), the following results occur: mon_grouping Formatted Value 3;-1 123456'789 3 123'456'789 3;2;-1 1234'56'789 3;2 12'34'56'789
positive_sign	Specifies the string used to indicate a nonnegative-formatted monetary quantity.
negative_sign	Specifies the string used to indicate a negative-formatted monetary quantity.
int_frac_digits	Specifies an integer value representing the number of fractional digits (those after the decimal delimiter) to be displayed in a formatted monetary quantity using the int_curr_symbol value.

†The current implementation of the Compaq C Run-Time Library allows more than four characters to be specified. However, the user should not rely on this fact and use it exactly as specified. The 4-character limit will be implemented in a future version of the Compaq C Run-Time Library.

(continued on next page)

Table 2–4 (Cont.) LC_MONETARY Category Keywords

Keyword	Description
frac_digits	Specifies an integer value representing the number of fractional digits (those after the decimal delimiter) to be displayed in a formatted monetary quantity using the currency_symbol value.
p_cs_precedes	Specifies an integer value indicating whether the int_curr_symbol or currency_symbol string precedes or follows the value for a nonnegative-formatted monetary quantity. The following integer values are recognized: 0 The currency symbol follows the monetary quantity. 1 The currency symbol precedes the monetary quantity.
p_sep_by_space	Specifies an integer value indicating whether the int_curr_symbol or currency_symbol string is separated by a space from a nonnegative-formatted monetary quantity. The following integer values are recognized: 0 No space separates the currency symbol from the monetary quantity. 1 A space separates the currency symbol from the monetary quantity. 2 A space separates the currency symbol and the positive_sign string, if adjacent.
n_cs_precedes	Specifies an integer value indicating whether the int_curr_symbol or currency_symbol string precedes or follows the value for a negative-formatted monetary quantity. The following integer values are recognized: 0 The currency symbol follows the monetary quantity. 1 The currency symbol precedes the monetary quantity.
n_sep_by_space	Specifies an integer value indicating whether the int_curr_symbol or currency_symbol string is separated by a space from a negative-formatted monetary quantity. The following integer values are recognized: 0 No space separates the currency symbol from the monetary quantity. 1 A space separates the currency symbol from the monetary quantity. 2 A space separates the currency symbol and the negative_sign string, if adjacent.

(continued on next page)

Table 2–4 (Cont.) LC_MONETARY Category Keywords

Keyword	Description
p_sign_posn	Specifies an integer value indicating the positioning of the positive_sign string for a nonnegative-formatted monetary quantity. The following integer values are recognized: 0 A left parenthesis and right parenthesis symbol enclose both the monetary quantity and the int_curr_symbol or currency_symbol string. 1 The positive_sign string precedes the quantity and the int_curr_symbol or currency_symbol string. 2 The positive_sign string follows the quantity and the int_curr_symbol or currency_symbol string. 3 The positive_sign string immediately precedes the int_curr_symbol or currency_symbol string. 4 The positive_sign string immediately follows the int_curr_symbol or currency_symbol string.
n_sign_posn	Specifies an integer value indicating the positioning of the negative_sign string for a negative-formatted monetary quantity. The following integer values are recognized: 0 A left parenthesis and right parenthesis symbol enclose both the monetary quantity and the int_curr_symbol or currency_symbol string. 1 The negative_sign string precedes the quantity and the int_curr_symbol or currency_symbol string. 2 The negative_sign string follows the quantity and the int_curr_symbol or currency_symbol string. 3 The negative_sign string immediately precedes the int_curr_symbol or currency_symbol string. 4 The negative_sign string immediately follows the int_curr_symbol or currency_symbol string.

2.5.2 Monetary Format Variations

You can produce a unique customized monetary format by changing the value of a single statement. Table 2–5 shows the results of using all combinations of defined values for the p_cs_precedes, p_sep_by_space, and p_sign_posn statements.

Table 2–5 Monetary Format Variations

	p_sep_by_space =	2	1	0
p_cs_precedes = 1	p_sign_posn = 0	($1.25)	($ 1.25)	($1.25)
	p_sign_posn = 1	+ $1.25	+$ 1.25	+$1.25
	p_sign_posn = 2	$1.25 +	$ 1.25+	$1.25+
	p_sign_posn = 3	+ $1.25	+$ 1.25	+$1.25
	p_sign_posn = 4	$ +1.25	$+ 1.25	$+1.25

(continued on next page)

Table 2–5 (Cont.) Monetary Format Variations

	p_sep_by_space =	2	1	0
p_cs_precedes = 0	p_sign_posn = 0	(1.25 $)	(1.25 $)	(1.25$)
	p_sign_posn = 1	+1.25 $	+1.25 $	+1.25$
	p_sign_posn = 2	1.25$ +	1.25 $+	1.25$+
	p_sign_posn = 3	1.25+ $	1.25 +$	1.25+$
	p_sign_posn = 4	1.25$ +	1.25 $+	1.25$+

The following is a sample LC_MONETARY category specified in a locale definition source file:

```
LC_MONETARY
#
int_curr_symbol         "<U><S><D><space>"
currency_symbol         "<dollar-sign>"
mon_decimal_point       "<period>"
mon_thousands_sep       "<comma>"
mon_grouping            3
positive_sign           "<plus-sign>"
negative_sign           "<hyphen>"
int_frac_digits         2
frac_digits             2
p_cs_precedes           1
p_sep_by_space          2
n_cs_precedes           1
n_sep_by_space          2
p_sign_posn             3
n_sign_posn             3
#
END LC_MONETARY
```

2.6 LC_NUMERIC Category

The LC_NUMERIC category defines rules and symbols for formatting nonmonetary numeric information. This category begins with the LC_NUMERIC and ends with the END LC_NUMERIC trailer.

All operands for the LC_NUMERIC category keywords are defined as string or integer values. String values are bounded by double quotation marks ("). All values are separated from the keyword they define by one or more blank characters (spaces or tabs). Two adjacent double quotation characters ("") indicate an undefined string value. A negative one (-1) indicates an undefined integer value.

Table 2–6 lists the statement keywords recognized in the LC_NUMERIC category.

Table 2–6 LC_NUMERIC Category Keywords

Keyword	Description
copy	Specifies the name of an existing locale to be used as the definition of this category. If you specify a copy statement, you cannot specify any other keyword.
decimal_point	Specifies the decimal delimiter string used to format nonmonetary numeric quantities. This keyword cannot be omitted and cannot be set to the undefined string value.
thousands_sep	Specifies the string separator used for grouping digits to the left of the decimal delimiter in formatted nonmonetary numeric quantities.
grouping	Defines the size of each group of digits in formatted monetary quantities. The operand for the grouping keyword consists of a sequence of integers separated by semicolons. Each integer specifies the number of digits in a group. The first integer defines the size of the group immediately to the left of the decimal delimiter. Subsequent integers define succeeding groups to the left of the previous group. Grouping is performed for each integer specified for the grouping keyword. If the last integer is not -1, it is used repeatedly to group any remaining digits. If the last integer is -1, no more grouping is performed. A sample interpretation of the grouping statement follows. Assuming a value of 123456789 to be formatted and a thousands_sep operand of ' (single quotation mark), the following results occur: grouping — Formatted Value 3;-1 — 123456'789 3 — 123'456'789 3;2;-1 — 1234'56'789 3;2 — 12'34'56'789

The following is a sample LC_NUMERIC category specified in a locale definition source file:

```
LC_NUMERIC
#
decimal_point   "<period>"
thousands_sep   "<comma>"
grouping        <3>
#
END LC_NUMERIC
```

2.7 LC_TIME Category

The LC_TIME category defines rules and symbols for formatting time and date information. This category begins with the LC_TIME category header and ends with the END LC_TIME trailer.

All operands for the LC_TIME category keywords are defined as string or integer values. String values are bounded by double quotation marks ("). All values are separated from the keyword they define by one or more blank characters (spaces or tabs). Two adjacent double quotation characters ("") indicate an undefined string value. Field descriptors, described later in this section, are used by commands and subroutines that query the LC_TIME category to represent elements of time and date formats.

2.7.1 Keywords

Table 2–7 lists the statement keywords recognized in the LC_TIME category.

Table 2–7 LC_TIME Category Keywords

Keyword	Description
copy	Specifies the name of an existing locale to be used as the definition of this category.
	If you specify a copy statement, you cannot specify any other keyword.
abday	Defines the abbreviated weekday names corresponding to the %a field descriptor.
	Recognized values consist of seven strings separated by semicolons. The first string corresponds to the abbreviated name for the first day of the week (Sun), the second to the abbreviated name for the second day of the week, and so on.
day	Defines the full spelling of the weekday names corresponding to the %A field descriptor.
	Recognized values consist of seven strings separated by semicolons. The first string corresponds to the full spelling of the name of the first day of the week (Sunday), the second to the name of the second day of the week, and so on.
abmon	Defines the abbreviated month names corresponding to the %b field descriptor.
	Recognized values consist of 12 strings separated by semicolons. The first string corresponds to the abbreviated name for the first month of the year (Jan), the second to the abbreviated name for the second month of the year, and so on.
mon	Defines the full spelling of the month names corresponding to the %B field descriptor.
	Recognized values consist of 12 strings separated by semicolons. The first string corresponds to the full spelling of the name for the first month of the year (January), the second to the full spelling of the name for the second month of the year, and so on.
d_t_fmt	Defines the string used for the standard date-and-time format corresponding to the %c field descriptor. The string can contain any combination of characters and field descriptors.
d_fmt	Defines the string used for the standard date format corresponding to the %x field descriptor. The string can contain any combination of characters and field descriptors.
t_fmt	Defines the string used for the standard time format corresponding to the %X field descriptor. The string can contain any combination of characters and field descriptors.

(continued on next page)

Table 2–7 (Cont.) LC_TIME Category Keywords

Keyword	Description
am_pm	Defines the strings used to represent a.m. (before noon) and p.m. (afternoon) corresponding to the %p field descriptor. Recognized values consist of two strings separated by semicolons. The first string corresponds to the a.m. designation, the second string corresponds to the p.m. designation.
t_fmt_ampm	Defines the string used for the standard 12-hour time format that includes an am_pm value (%p field descriptor). This statement corresponds to the %r field descriptor. The string can contain any combination of characters and field descriptors. If the string is empty, the 12-hour format is not supported by the locale.
era	Defines how the years are counted and displayed for each era in a locale, corresponding to the %E field descriptor modifier. For each era, there must be one string in the following format: *direction*:*offset*:*start_date*:*end_date*:*name*:*format* The variables for the era string format are defined as follows: • *direction* — Specifies a minus (-) or a plus (+) character. The minus character (-) indicates that years count in the negative direction when moving from the start date to the end date. The plus character (+) indicates that years count in the positive direction when moving from the start date to the end date. • *offset* — Specifies a number representing the first year of the era corresponding to the %Ey field descriptor. • *start_date* — Specifies the starting date of the era in yyyy/mm/dd format, where yyyy, mm, and dd are the year, month, and day, respectively, on the Gregorian calendar. Years prior to the year A.D. 1 are represented as negative numbers. For example, an era beginning March 5 in the year 100 B.C. would be represented as -100/03/05. • *end_date* — Specifies the ending date of the era in the same form used for the start_date variable or one of the two special values -* or +*. A -* value indicates that the ending date of the era extends backward to the beginning of time. A +* value indicates that the ending date of the era extends forward to the end of time. Therefore, the ending date can be chronologically before or after the starting date of the era. For example, the strings for the Christian eras A.D. and B.C. would be entered, respectively, in the following way: +:0:0000/01/01:+*:AD:%Ey %EC +:1:-0001/12/31:-*:BC:%Ey %EC • *name* — Specifies a string representing the name of the era that is substituted for the %EC field descriptor. • *format* — Specifies a strftime, strptime, and wcsftime format string to use when formatting the %EY field descriptor. This string can contain any strftime, strptime, and wcsftime format control characters (except %EY) and locale-dependent multibyte characters.

(continued on next page)

Table 2–7 (Cont.) LC_TIME Category Keywords

Keyword	Description
	An era value consists of one string (enclosed in quotation marks) for each era. If more than one era is specified, each era string is separated by a semicolon (;).
era_d_fmt	Defines the string used to represent the date in alternate-era format corresponding to the %Ex field descriptor. The string can contain any combination of characters and field descriptors.
era_t_fmt	Defines the locale's alternative time format as represented by the %EX field descriptor for strftime, strptime, and wcsftime.
era_d_t_fmt	Defines the locale's alternative date-and-time format as represented by the %Ec field descriptor for strftime, strptime, and wcsftime.
alt_digits	Defines alternate strings for digits corresponding to the %O field descriptor. Recognized values consist of a group of strings separated by semicolons. The first string represents the alternate string for 0 (zero), the second string represents the alternate string for 1, and so on. You can specify a maximum of 100 alternate strings.

2.7.2 Field Descriptors

The LC_TIME locale definition source file uses field descriptors to represent elements of time and date formats. You can combine these field descriptors to create other field descriptors or to create time and date format strings. When used in format strings that contain field descriptors and other characters, field descriptors are replaced by their current values. All other characters are copied without change. Table 2–8 lists the field descriptors used by commands and subroutines that query the LC_TIME category for time formatting.

Table 2–8 LC_TIME Locale Field Descriptors

Field Descriptor	Meaning
%a	Represents the abbreviated weekday name (for example, Sun) defined by the abday statement.
%A	Represents the full weekday name (for example, Sunday) defined by the day statement.
%b	Represents the abbreviated month name (for example, Jan) defined by the abmon statement.
%B	Represents the full month name (for example, January) defined by the mon statement.
%c	Represents the date-and-time format defined by the d_t_fmt statement.
%C	Represents the century as a decimal number (00 to 99).
%d	Represents the day of the month as a decimal number (01 to 31).
%D	Represents the date in %m/%d/%y format (for example, 01/31/91).

(continued on next page)

Table 2–8 (Cont.) LC_TIME Locale Field Descriptors

Field Descriptor	Meaning
%e	Represents the day of the month as a decimal number (1 to 31). If the day of the month is not a 2-digit number, the leading digit is filled with a space character.
%Ec	Specifies the alternate date-and-time representation for the locale.
%EC	Specifies the name of the base year (period) in the locale's alternate representation.
%Ex	Specifies the alternate date representation for the locale.
%Ey	Specifies the offset from %EC (year only) in the locale's alternate representation.
%EY	Specifies the full alternate year representation.
%h	Represents the abbreviated month name (for example, Jan) defined by the `abmon` statement. This field descriptor is a synonym for the %b field descriptor.
%H	Represents the 24-hour clock hour as a decimal number (00 to 23).
%I	Represents the 12-hour clock hour as a decimal number (01 to 12).
%j	Represents the day of the year as a decimal number (001 to 366).
%m	Represents the month of the year as a decimal number (01 to 12).
%M	Represents the minutes of the hour as a decimal number (00 to 59).
%n	Specifies a new-line character.
%Od	Specifies the day of the month by using the locale's alternate numeric symbols.
%Oe	Specifies the day of the month by using the locale's alternate numeric symbols.
%OH	Specifies the hour (24-hour clock) by using the locale's alternate numeric symbols.
%OI	Specifies the hour (12-hour clock) by using the locale's alternate numeric symbols.
%Om	Specifies the month by using the locale's alternate numeric symbols.
%OM	Specifies the minutes by using the locale's alternate numeric symbols.
%OS	Specifies the seconds by using the locale's alternate numeric symbols.
%OU	Specifies the week number of the year (with Sunday as the first day of the week) by using the locale's alternate numeric symbols.
%Ow	Specifies the weekday as a number in the locale's alternate representation (Sunday = 0).
%OW	Specifies the week number of the year (with Monday as the first day of the week) by using the locale's alternate numeric symbols.
%Oy	Specifies the year (offset from %C) using the locale's alternate numeric symbols.
%p	Represents the a.m. or p.m. string defined by the `am_pm` statement.
%r	Represents the 12-hour clock time with a.m./p.m. notation as defined by the `t_fmt_ampm` statement.
%S	Represents the seconds of the minute as a decimal number (00 to 59).

(continued on next page)

Table 2–8 (Cont.) LC_TIME Locale Field Descriptors

Field Descriptor	Meaning
%t	Specifies a tab character.
%T	Represents 24-hour clock time in the format %H:%M:%S (for example, 16:55:15).
%U	Represents the week of the year as a decimal number (00 to 53). Sunday, or its equivalent as defined by the day statement, is considered the first day of the week for calculating the value of this field descriptor.
%w	Represents the day of the week as a decimal number (0 to 6). Sunday, or its equivalent as defined by the day statement, is considered to be 0 (zero) for calculating the value of this field descriptor.
%W	Represents the week of the year as a decimal number (00 to 53). Monday, or its equivalent as defined by the day statement, is considered the first day of the week for calculating the value of this field descriptor.
%x	Represents the date format defined by the d_fmt statement.
%X	Represents the time format defined by the t_fmt statement.
%y	Represents the year of the century (00 to 99).
%Y	Represents the year as a decimal number (for example, 1989).
%%	Specifies a % (percent sign) character.

2.7.3 Sample Locale Definition

The following is a sample LC_TIME category specified in a locale definition source file:

```
LC_TIME
#
#Abbreviated weekday names (%a)
abday   "<S><u><n>";"<M><o><n>";"<T><u><e>";"<W><e><d>";\
        "<T><h><u>";"<F><r><i>";"<S><a><t>"

#Full weekday names (%A)
day     "<S><u><n><d><a><y>";"<M><o><n><d><a><y>";\
        "<T><u><e><s><d><a><y>";"<W><e><d><n><e><s><d><a><y>";\
        <T><h><u><r><s><d><a><y>";"<F><r><i><d><a><y>";\
        <S><a><t><u><r><d><a><y>"

#Abbreviated month names (%b)
abmon   "<J><a><n>";"<F><e><b>";"<M><a><r>";"<A><P><r>";\
        "<M><a><y>";"<J><u><n>";"<J><u><l>";"<A><u><g>";\
        <S><e><P>";"<O><c><t>";"<N><o><v>";"<D><e><c>"

#Full month names (%B)
mon     "<J><a><n><u><a><r><y>";"<F><e><b><r><u><a><r><y>";\
        "<M><a><r><c><h>";"<A><P><r><i><l>";"<M><a><y>";\
        <J><u><n><e>";"<J><u><l><y>";"<A><u><g><u><s><t>";\
        "<S><e><P><t><e><m><b><e><r>";"<O><c><t><o><b><e><r>";\
        <N><o><v><e><m><b><e><r>";"<D><e><c><e><m><b><e><r>"

#Date-and-time format (%c)
#Note that for improved readability, this section uses actual
#characters, rather than symbolic names, and is inconsistent with
#the other sections in this example.  This is bad form.
#In practice, symbolic names should be used.
d_t_fmt         "%a %b %d %H:%M:%S %Y"
```

```
#
#Date format (%x)
d_fmt           "%m/%d/%y"
#
#Time format (%X)
t_fmt           "%H:%M:%S"
#
#Equivalent of AM/PM (%p)
am_pm           "<A><M>";"<P><M>"
#
#12-hour time format (%r)
#Note that for improved readability, this section uses actual
#characters, rather than symbolic names, and is inconsistent with
#the other sections in this example.  This is bad form.
#In practice, symbolic names should be used.
t_fmt_ampm      "%I:%M:%S %p"
#
era             "+:0:0000/01/01:+*:AD:%Ey %EC";\
"+:1:-0001/12/31:-*:BC:%Ey %EC"

era_d_fmt     ""
alt_digits    "<0><t><h>";"<1><s><t>";"<2><n><d>";"<3><r><d>";\
              "<4><t><h>";"<5><t><h>";"<6><t><h>";"<7><t><h>";\
              "<8><t><h>";"<9><t><h>";"<1><0><t><h>"
#
END LC_TIME
```

3

Character Set Description (Charmap) File

The character set description file, called the charmap file, defines character symbols as character encodings. This file is the source file for a coded character set, or codeset.

3.1 Portable Character Set

All supported codesets have the Portable Character Set (PCS) as a proper subset. The PCS consists of the character symbols (listed by their standardized symbolic names) and their hexadecimal encodings, as shown in Table 3–1.

Table 3–1 Portable Character Set

Symbol Name	Hexadecimal Encoding
<NUL>	\x00
<alert>	\x07
<backspace>	\x08
<tab>	\x09
<newline>	\x0A
<vertical-tab>	\x0B
<form-feed>	\x0C
<carriage-return>	\x0D
<space>	\x20
<exclamation-mark>	\x21
<quotation-mark>	\x22
<number-sign>	\x23
<dollar-sign>	\x24
<percent>	\x25
<ampersand>	\x26
<apostrophe>	\x27
<left-parenthesis>	\x28
<right-parenthesis>	\x29
<asterisk>	\x2A
<plus-sign>	\x2B
<comma>	\x2C
<hyphen>	\x2D

(continued on next page)

Table 3–1 (Cont.) Portable Character Set

Symbol Name	Hexadecimal Encoding
<period>	\x2E
<slash>	\x2F
<zero>	\x30
<one>	\x31
<two>	\x32
<three>	\x33
<four>	\x34
<five>	\x35
<six>	\x36
<seven>	\x37
<eight>	\x38
<nine>	\x39
<colon>	\x3A
<semi-colon>	\x3B
<less-than>	\x3C
<equal-sign>	\x3D
<greater-than>	\x3E
<question-mark>	\x3F
<commercial-at>	\x40
<A>	\x41
<B>	\x42
<C>	\x43
<D>	\x44
<E>	\x45
<F>	\x46
<G>	\x47
<H>	\x48
<I>	\x49
<J>	\x4A
<K>	\x4B
<L>	\x4C
<M>	\x4D
<N>	\x4E
<O>	\x4F
<P>	\x50
<Q>	\x51
<R>	\x52
<S>	\x53

(continued on next page)

Table 3–1 (Cont.) Portable Character Set

Symbol Name	Hexadecimal Encoding
<T>	\ x54
<U>	\ x55
<V>	\ x56
<W>	\ x57
<X>	\ x58
<Y>	\ x59
<Z>	\ x5A
<left-bracket>	\ x5B
<backslash>	\ x5C
<right-bracket>	\ x5D
<circumflex>	\ x5E
<underscore>	\ x5F
<grave-accent>	\ x60
<a>	\ x61
<b>	\ x62
<c>	\ x63
<d>	\ x64
<e>	\ x65
<f>	\ x66
<g>	\ x67
<h>	\ x68
<i>	\ x69
<j>	\ x6A
<k>	\ x6B
<l>	\ x6C
<m>	\ x6D
<n>	\ x6E
<o>	\ x6F
<p>	\ x70
<q>	\ x71
<r>	\ x72
<s>	\ x73
<t>	\ x74
<u>	\ x75
<v>	\ x76
<w>	\ x77
<x>	\ x78
<y>	\ x79

(continued on next page)

Table 3–1 (Cont.) Portable Character Set

Symbol Name	Hexadecimal Encoding
<z>	\ x7A
<left-brace>	\ x7B
<vertical-line>	\ x7C
<right-brace>	\ x7D
<tilde>	\ x7E

3.2 Components of a Charmap File

A charmap file has the following components:

- An optional special symbolic name declarations section

  Each declaration in this section consists of a special symbolic name, followed by one or more space or tab characters, and a value. The following list describes the special symbolic names that you can include in the declarations section:

  `<code_set_name>`

  Specifies the name of the codeset for which the charmap file is defined. This value determines the value returned by the `nl_langinfo (CODESET)` subroutine. If <code_set_name> is not declared, the name for the Portable Character Set is used.

  `<mb_cur_max>`

  Specifies the maximum number of bytes in a character for the codeset. Valid values are 1 to 4. The default value is 1.

  `<mb_cur_min>`

  Specifies the minimum number of bytes in a character for the codeset. Since all supported codesets have the Portable Character Set as a proper subset, this value must be 1.

  `<escape_char>`

  Specifies the escape character that indicates encodings in hexadecimal or octal notation. The default value is a backslash (\ ).

  `<comment_char>`

  Specifies the character used to indicate a comment within a charmap file. The default value is the number sign (#).

- The CHARMAP section header

  This header marks the beginning of the section that associates character symbols with encodings.

- Mapping statements for characters in the codeset

  Each statement specifies a symbolic name for a character and the associated encoding for that character. A mapping statement has the following format:

  <char_symbol> encoding

A symbolic name begins with the left angle-bracket (<) character and ends with the right angle-bracket (>) character. For *char_symbol* (the name between < and >), you can use any characters from the Portable Character Set, except for control and space characters. You can use a > in *char_symbol*; if you do, precede all > characters except the last one with the escape character (as specified by the <escape_char> special symbolic name).

An encoding is specified as one or more character constants, with the maximum number of character constants specified by the <mb_cur_max> special symbolic name. The encoding may be specified as decimal, octal, or hexadecimal constants with the following formats:

- Decimal constant: \ d*nn* or \ d*nnn*, where *n* is any decimal digit
- Octal constant: \ *nn* or \ *nnn*, where *n* is any octal digit
- Hexadecimal constant: \ x*nn*, where *n* is any hexadecimal digit

The following are sample character symbol definitions:

```
<A>         \d65        #decimal constant
<B>         \x42        #hexadecimal constant
<j10101>    \x81\xA1    #multiple hexadecimal constants
```

You can also define a range of symbolic names and corresponding encoded values, where the nonnumeric prefix for each symbolic name is common, and the numeric portion of the second symbolic name is equal to or greater than the numeric portion of the first symbolic name. In this format, a symbolic name value consists of zero or more nonnumeric characters followed by an integer of one or more decimal digits. This format defines a series of symbolic names. For example, the string <j0101>...<j0104> is interpreted as the symbolic names <j0101>, <j0102>, <j0103>, and <j0104>, in that order.

In statements defining ranges of symbolic names, the specified encoded value is the value for the first symbolic name in the range. Subsequent symbolic names have encoded values in increasing order. Consider the following sample statement:

```
<j0101>...<j0104>        \d129\d254
```

This sample statement is interpreted as follows:

```
<j0101> \d129\d254
<j0102> \d129\d255
<j0103> \d130\d0
<j0104> \d130\d1
```

You cannot assign multiple encodings to one symbolic name, but you can create multiple names for one encoded value because some characters have several common names. For example, the . character is called a period in some parts of the world, and a full stop in others. You can specify both names in the charmap. For example:

```
<period>        \x2e
<full-stop>     \x2e
```

Any comments must begin with the character specified by the <comment_char> special symbolic name. When an entire line is a comment, you must specify the <comment_char> in the first column of the line.

- The END CHARMAP section trailer

  This trailer indicates the end of character map statements.

The following is a portion of a sample charmap file:

```
CHARMAP
<code_set_name>         "ISO8859-1"
<mb_cur_max>            1
<mb_cur_min>            1
<escape_char>           \
<comment_char>          #

<NUL>                   \x00
<SOH>                   \x01
<STX>                   \x02
<ETX>                   \x03
<EOT>                   \x04
<ENQ>                   \x05
<ACK>                   \x06
<alert>                 \x07
<backspace>             \x08
<tab>                   \x09
<newline>               \x0a
<vertical-tab>          \x0b
<form-feed>             \x0c
<carriage-return>       \x0d
END CHARMAP
```

4

Command Reference

This section describes the following commands offered by the Compaq C Run-Time Library utilities:

- GENCAT
- ICONV COMPILE
- ICONV CONVERT
- LOCALE COMPILE
- LOCALE LOAD
- LOCALE UNLOAD
- LOCALE SHOW CHARACTER_DEFINITIONS
- LOCALE SHOW CURRENT
- LOCALE SHOW PUBLIC
- LOCALE SHOW VALUE
- zic

GENCAT

Merges message text source files into a message catalog file.

Format

GENCAT *msgfile*[,...] *catfile*

Parameters

msgfile
Required.

Name of the message text source file. The default file type is .MSGX.

catfile
Required.

Name of the message catalog output file. If `catfile` already exists, a new version is created that includes the messages in the existing catalog. The file type must be .CAT.

Qualifiers

None.

Description

The GENCAT command creates new message catalogs from one or more input source files and an existing catalog file (if one exists). A message catalog is a binary file containing the messages for an application. This includes all messages that the application issues, such as error messages, screen displays, and prompts. Applications retrieve messages from a message catalog using the `catopen`, `catgets`, and `catclose` C Run-Time Library routines. See the *Compaq C Run-Time Library Reference Manual for OpenVMS Systems* for details of these routines.

A message text source file is a text file that you create to hold messages printed by your program. You can use any text editor to enter messages into the text source file. Messages can be grouped into sets, usually to represent functional subsets of your program. Each message has a numeric identifier, which must be unique within its set. The message text source file can also contain commands recognized by GENCAT for manipulating sets and individual messages.

You can specify any number of message text source files. The GENCAT command processes multiple source files one after the other in the sequence that you specify them. Each successive source file modifies the catalog.

If a message catalog with the name *catfile* exists, GENCAT creates a new version of the file that includes the contents of the older version and then modifies it. If the catalog does not exist, GENCAT creates the catalog with the name *catfile*.

The *catfile* can contain the following commands:

- *message_number text*

Inserts *text* as a message with the identifier *message_number*. Follow these guidelines:

- Numbers must be ascending within each set. You can skip a number, but you cannot go back to add a missing number or replace an existing number during a GENCAT session.
- If the message text is empty and a space or tab field separator is present, an empty string is stored in the message catalog.
- If a message source line has a message number but neither a field separator nor message text, the existing message with that number (if any) is deleted from the catalog.

- $delset *set_number*

  Deletes the set of messages indicated by *set_number*.

- $quote *character*

  Sets the quote character to *character*. See the Examples section for more information.

- $set *set_number*

  Indicates that all messages entered after this command are placed in the set indicated by *set_number*. You can change the set by entering another $set command. However, set numbers must be entered in ascending order; you cannot go back to a lower numbered set during the GENCAT session. If the command is not used, the default set number is 1.

Each initial keyword or number must be followed by white space. The GENCAT utility ignores any line that begins with a space, a tab, or a dollar sign ($) character followed by a space, a tab, or a newline character. Therefore, you can use these sequences to start comments in your *catfile*. Blank lines are also ignored. Finally, you can place comments on the same line after the $delset, $quote, or $set commands because GENCAT ignores anything that follows these commands.

A line beginning with a digit marks a message to be included in the catalog. You can specify any amount of white space between the message ID number and the message text; however, when the message text is not delimited by quotation marks, one space or tab character is recommended. When message text is not in quotation marks, GENCAT treats additional white space as part of the message. When message text is enclosed in quotation marks, GENCAT ignores all spaces or tabs between the message ID and the first quotation character.

Escape sequences such as those recognized by the C language can be used in text. The escape character (\), a backslash, can be used to insert special characters in the message text. See Table 4–1.

Table 4–1 GENCAT Command: Special Characters

Escape Sequence	Character
\ n	New Line
\ t	Horizontal Tab
\ v	Vertical Tab
\ b	Backspace
\ r	Carriage Return
\ f	Form Feed
\ \	Backslash Character (\ ). Use to continue message text on the following line.
\ *ddd*	The single-byte character associated with the octal value *ddd*. You can specify one, two, or three octal digits. However, you must include leading zeros if the characters following the octal digits are also valid octal digits; for example, the octal value for $ (dollar sign) is 44. To insert $5.00 into a message, use \ 0445.00, not \ 445.00; otherwise the 5 is parsed as part of the octal value.

Error

When GENCAT reports an error, no action is taken on any commands and an existing catalog is left unchanged.

Examples

1. `$set 10 Communication Error Messages`

   This example uses the $set command in a source file to assign a set number to a group of messages.

   The message set number is 10. All messages after the $set command and up to the next $set command are assigned a message set number of 10. (Set numbers must be assigned in ascending order but they need not be contiguous.)

   You can include a comment in the $set command.

2. `$delset 10 Communication Error Messages`

   This example uses the $delset command to remove from a catalog all messages belonging to the specified message set (10, in this case).

   The $delset command must be placed in the proper set number order with respect to any $set commands in the same source file. You can include a comment in the $delset command.

3. `12 "file removed"`

   This example shows how to enter the message text and assign a message ID number to it. In this case, a message ID of 12 is assigned to the text that follows it.

   Leave at least one space or tab character between the message ID number and the message text, but you can include more spaces or tabs if you prefer. If you do include more spaces or tabs, they are ignored when the message text is in quotation marks and they are considered part of the text when the message text is not in quotation marks.

   Message numbers must be in ascending order within a single message set but they need not be contiguous.

   All text following the message number and up to the end of the line is included as message text. If you place the escape character (\), a backslash, as the last character on the line, the message text continues on the following line. Consider the following example:

   ```
   This is the text associated with \
   message number 5.
   ```

   The two lines in the example define the following single-line message:

   `This is the text associated with message number 5.`

4. ```
   $quote "   Use a double quote to delimit message text
   $set 10            Message Facility - Quote command messages
   1 "Use the $quote command to define a character \
   \n for delimiting message text" \n
   2 "You can include the \"quote\" character in a message \n \
   by placing a \\ (backslash) in front of it" \n
   3 You can include the "quote" character in a message \n \
   by having another character as the first nonspace \
   \n character after the message ID number \n
   $quote
   4 You can disable the quote mechanism by \n \
   using the $quote command without \n a character \
   after it \n
   ```

   This example shows the effect of a quote character.

   The $quote command defines the double quote (") as the quote character. The quote character must be the first nonspace character after the message number. Any text following the next occurrence of the quote character is ignored.

   This example also shows two ways to include the quote character in the message text:

   - Place a backslash (\) in front of the quote character.
   - Use another character as the first nonspace character after the message number. This disables the quote character for that message only.

   This example also shows the following:

   - A backslash (\) is still required to split a quoted message across lines.
   - To display a backslash (\) in a message, you must place another backslash (\) in front of it.
   ```

- You can format your message with a new-line character by using \ n.
- If you use the $quote command with no character argument, you disable the quote mechanism.

ICONV COMPILE

Creates a conversion table file from a conversion source file. The conversion table file is used by the ICONV CONVERT command to convert characters in a file from one codeset to another.

Format

ICONV COMPILE *sourcefile tablefile*

Parameters

sourcefile
Required.

Name of the conversion source file. The default file type is .ISRC. The file naming convention that Compaq uses for conversion source files is:

```
fromcodeset_tocodeset.isrc
```

tablefile
Required.

Name of the conversion table file to be created. The default file type is .ICONV. The required file naming convention for conversion table files is:

```
fromcodeset_tocodeset.iconv
```

Public conversion table files are in the directory defined by the logical name SYS$I18N_ICONV. Put new conversion table files in the same directory if you want to make them available systemwide.

Qualifiers

/LISTING[=*listfile*]
Directs ICONV COMPILE to produce a listing file, which contains the source file listing and any error messages generated during compilation. If the file name is omitted from the qualifier, the default listing file name is *sourcefile*.LIS.

Description

The ICONV commands support any 1- to 4-byte codesets that are state independent. They do not support state-dependent codesets.

Note

There is an implementation restriction in the *tocodeset* encodings in this implementation. The characters in *tocodeset* must not use 0XFF in the fourth byte.

The conversion source file contains the character conversion rules for a specific conversion.

The format of a codeset conversion source file is defined as follows:

```
<fromcodeset_mb_cur_max>    value
<fromcodeset_mb_cur_min>    value
<tocodeset_mb_cur_max>      value
<tocodeset_mb_cur_min>      value
<fallback_code>             value
<escape_char>               value
<comment_char>              value
<fromcodeset_range>         value...value;value...value;...;value...value
ICONV_TABLE
fromvalue                         tovalue
fromvalue                         tovalue
   .                                 .
   .                                 .
   .                                 .
fromvalue                         tovalue
END ICONV_TABLE
```

where the <...> symbols and their associated values are codeset declarations, and the *fromvalue*/*tovalue* pairs are character conversion rules.

Codeset Declarations

The codeset declarations must precede the character conversion rules. Each declaration consists of a symbol, starting in column 1 and including the surrounding brackets, followed by one or more blanks (tabs or spaces), followed by the value to be assigned to the symbol. See Table 4–2.

Table 4–2 Codeset Declarations

Symbol	Value
<fromcodeset_mb_cur_max>	The maximum number of bytes in a character in the fromcodeset. This value defaults to 1.
<fromcodeset_mb_cur_min>	The minimum number of bytes in a character in the fromcodeset. This value must be less than or equal to fromcodeset_mb_cur_max. If this value is not specified, it defaults to the value of fromcodeset_mb_cur_max.
<tocodeset_mb_cur_max>	The maximum number of bytes in a character in the tocodeset. This value defaults to 1.
<tocodeset_mb_cur_min>	The minimum number of bytes in a character in the tocodeset. This value must be less than or equal to tocodeset_mb_cur_max. If this value is not specified, it defaults to the value of tocodeset_mb_cur_max.

(continued on next page)

Table 4–2 (Cont.) Codeset Declarations

Symbol	Value
<fallback_code>	The *tovalue*s for the *fromvalue*s that appear in the <fromcodeset_range> but are not specified between ICONV_TABLE and END ICONV_ TABLE. Specify one of three kinds of values: • SAME — Specifies that the *tovalue*s are the same as the *fromvalue*s. • ERROR — Specifies that the conversion from the *fromvalue* to a *tovalue* is not supported. ICONV CONVERT issues a warning and ignores the rest of the record read. The Compaq C Run-Time Library routine iconv returns to the caller with an "illegal character" error. • User-defined *tovalue* — The *fromvalue*s are converted to the specified user-defined *tovalue*. The user-defined *tovalue* can represent a multibyte character with the restriction that 0XFF cannot be used as the value in the fourth byte. The settings for user-defined *tovalue*s for <fallback_code> are the same as the settings for character conversion rule values. You can use octal, decimal, or hexadecimal digits. If the <fallback_code> is not specified, it defaults to SAME.
<escape_char>	The escape character used to indicate that subsequent characters are interpreted in a special way. The escape character defaults to backslash (\).
<comment_char>	The character that, when placed in column 1 of a line, indicates that the line will be ignored. The default comment character is the number sign (#).
<fromcodeset_range>	The fromcodeset encoding ranges. Specify this declaration if the fromcodeset is a multibyte codeset. If the fromcodeset is omitted, it defaults to a single-byte codeset and the table created by ICONV COMPILE will support only single-byte fromcodeset conversions.

When specifying codeset encoding ranges for the fromcodeset, every zone of characters must be specified. If any zones of characters are missing from the <fromcodeset_range> specification, the codeset conversion might be incorrect. It is very important to specify the codeset encoding ranges correctly for the fromcodesets supported by the rest of the Compaq C Run-Time Library. If this is not done, the codeset support for iconv and the rest of the Compaq C Run-Time Library will not be consistent.

For example, the fromcodeset ranges for EUCJP are specified as:

```
<fromcodeset_range>  \x0...\x7f;\x8e\xa1...\x8e\xfe;
                     \xa1\xa1...\xfe\xfe;\x8f\xa1\xa1...\x8f\xfe\xfe
```

The settings for <fromcodeset_range> values are the same as the settings for character conversion rule values. You can use octal, decimal, or hexadecimal digits.

Character Conversion Rules

The character conversion rules are all the lines between the string ICONV_TABLE starting in column 1 and END ICONV_TABLE starting in column 1.

Character conversion rules must begin in column 1.

Empty lines and lines containing a comment_char in the first column are ignored. Comments are optional.

Character conversion rules can have one of two forms:

fromvalue *tovalue*

fromvalue...fromvalue *tovalue*

Place one or more blanks (tabs or spaces) between *fromvalue* and *tovalue*.

Use the first format to define a single-character conversion rule. For example:

```
\d32       \d101
\d37       \d106
```

Use the second format to define a range of character conversion rules. In this format, the ending *fromvalue* must be equal to or greater than the starting *fromvalue*. The subsequent *fromvalue*s defined by the range are converted to *tovalue*s in increasing order.

For example, consider the following line:

```
\d223\d32...\d223\d35       \d129\d254
```

This line is interpreted as:

```
\d223\d32       \d129\d254
\d223\d33       \d129\d255
\d223\d34       \d130\d0
\d223\d35       \d130\d1
```

For settings of *fromvalue* and *tovalue*:

- A decimal constant is defined as one, two, or three decimal digits preceded by the escape character and lowercase d. For example: \d42.
- An octal constant is defined as one, two, or three octal digits preceded by the escape character. For example: \141.
- A hexadecimal constant is defined as one or two hexadecimal digits preceded by the escape character and a lowercase x. For example: \x6a.

Each constant represents a single-byte value. You can represent multibyte values by concatenating two or more decimal, octal, or hexadecimal constants.

Note

When constants are concatenated for multibyte values, they must have the same radix (decimal, octal, or hexadecimal). Only characters in the Portable Character Set can be used to construct conversion source files.

Also see the ICONV CONVERT command.

Errors

If an error is encountered during processing, ICONV COMPILE does not generate an output *tablefile*. If a warning is encountered, a valid table file is created. However, because a warning can indicate a user error, always check the returned warning messages.

Some ICONV COMPILE error messages and their descriptions follow.

`%ICONV-E-INVFCSRNG, syntax error in <fromcodeset_range> definition`

The previous error occurs when the definition of the <fromcodeset_range> symbol does not conform to the required syntax. The <fromcodeset_range> symbol defines encoding ranges and is required for multibyte codesets.

`%ICONV-E-INVSYNTAX, invalid file syntax`

The previous error occurs when a line in the source does not conform to the required syntax.

`%ICONV-E-BADTABLE, bad table caused by invalid value for <fromcodeset_range> definition`

The previous error occurs when an invalid value is specified for the codeset encoding ranges. The encoding ranges are defined by the <fromcodeset_range> symbol.

Examples

1. `$ ICONV COMPILE /LISTING EUCTW_DECHANYU.ISRC EUCTW_DECHANYU.ICONV`

   This example shows how to create a conversion table file to convert the EUCTW codeset to the DECHANYU codeset. The listing file, EUCTW_DECHANYU.LIS, contains a listing of the source file and any error messages generated by the compiler.

ICONV CONVERT

Converts characters in a file from one codeset to another codeset. The converted characters are written to an output file.

Format

ICONV CONVERT *infile outfile*

Parameters

infile
Required.

Name of the file that contains the characters to be converted. The /FROMCODE qualifier specifies the codeset of the characters in this file.

outfile
Required.

Name of the file created by ICONV CONVERT. The /TOCODE qualifier specifies the codeset of the characters in this file.

Qualifiers

/FROMCODE=*fromcodeset*
Required.

Specifies the codeset of the characters in *infile*.

/TOCODE=*tocodeset*
Required.

Specifies the codeset of the characters in *outfile*.

Description

The ICONV CONVERT command converts the characters in *infile* from the codeset identified by the /FROMCODE qualifier to the codeset identified by the /TOCODE qualifier. The converted file is written to *outfile*.

The conversion is done in one of two ways:

- Using a conversion table file to look up the converted characters. This is the default method. Conversion table files are created by the DCL command ICONV COMPILE.
- Using a shareable image file that implements the required conversion. This method can be used whenever the implementation of a converter by table is either not convenient, for example, huge virtual address space versus small space by algorithm, or not possible, for example, for state dependent encoding like ISO2022.

The converter's file naming convention, valid for both table or image file type of implementations, is:

```
fromcodeset_tocodeset.iconv
```

Note

If you add conversion files to your system, they must use the same file-naming convention.

ICONV CONVERT searches your current directory for a converter file. If it cannot find the file, it then searches the system directory defined by the logical name SYS$I18N_ICONV.

Examples

1.
```
$ ICONV CONVERT /FROMCODE=EUCTW /TOCODE=DECHANYU -
_$ FROMFILE.DAT TOFILE.DAT
```

   This example shows a conversion from EUCTW characters to DECHANYU characters. The EUCTW characters in the file FROMFILE.DAT are converted to the corresponding DECHANYU characters. The converted characters are stored in the file TOFILE.DAT.

LOCALE COMPILE

Converts a locale source file into a binary locale file. The binary locale file is used by those utilities and C routines that are dependent on the setting of the international environment logical names.

Format

LOCALE COMPILE *sourcefile*

Parameters

sourcefile
Required.

Name of the locale source file, which defines each category of the locale. The default file type for the source file is .LSRC. For the definition of the locale source file format, see Chapter 2.

Qualifiers

/CHARACTER_DEFINITIONS=*filename*
/NOCHARACTER_DEFINITIONS
Optional. Default: /NOCHARACTER_DEFINITIONS

Specifies a character-set description file (charmap) for the locale. This file maps characters to their actual character encodings.

If a charmap is not specified, no symbolic names (other than collating symbols defined in a collating symbol keyword) are allowed in the locale source file.

For a definition of the charmap file format, see Chapter 3. The default file type for a charmap is .CMAP.

/DISPLAY[=[NO]HOLE]
Optional. Default: /DISPLAY=NOHOLE

Used with certain Chinese locales and terminals to specify that 4-byte characters occupy four printing positions (columns) on the terminal display. The default value (/DISPLAY=NOHOLE) specifies that 4-byte characters occupy two printing positions.

/IGNORE=WARNINGS
/NOIGNORE
Optional. Default: /NOIGNORE

Generates an output file even if LOCALE COMPILE issues warning messages. Use the /IGNORE keyword cautiously because the warnings could indicate user errors that you might want to correct before using the resulting locale file.

/LISTING[=*filename*]
/NOLISTING
Optional. Batch default: /LISTING; interactive default: /NOLISTING

Name of the listing file. The /SHOW qualifier controls the information included in the listing file. If the file name is omitted, the default is *sourcefile*.LIS.

/OUTPUT=[*filename*]
/NOOUTPUT
Optional. Default: /OUTPUT=*sourcefile*.LOCALE

Name of the output file. Public locales are stored in the directory defined by the logical name SYS$I18N_LOCALE. If the output file is in any other location, the locale is private.

/NOOUTPUT results in no output file creation, even if the compilation succeeds.

/SHOW[=(*keyword*[,...])]
Optional. Default: /SHOW=(SOURCE,TERMINAL)

/SHOW, together with /LISTING, controls the information included in the listing file. You can specify the following *keywords*:

Keyword	Description
ALL	Include all information.
BRIEF	Include a summary of the symbol table.
[NO]CHARACTER_ DEFINITIONS	Include or omit the charmap file.
NONE	Do not print any information. The listing file contains only the generated error messages.
[NO]SOURCE	Include or omit a listing of the source file.
[NO]STATISTICS	Include or omit compiler performance information.
[NO]SYMBOLS	Include or omit a listing of the charmap symbol table.
[NO]TERMINAL	Display compiler messages at the terminal.

Description

Use the LOCALE COMPILE command to add new locales to your system in addition to those supplied by Compaq. To compile a locale, LOCALE COMPILE requires two files:

- A charmap file that defines the character set for the locale. If you do not specify a charmap file, symbolic names cannot be specified in the locale source file. If this happens, LOCALE COMPILE issues an error or warning message, depending on the category processed, and no output file is produced. (Also see the /IGNORE qualifier.)
- A locale source file. This file describes one or more of the locale categories: LC_CTYPE, LC_COLLATE, LC_MESSAGES, LC_MONETARY, LC_NUMERIC, and LC_TIME.

Errors

The following error messages are related to the LOCALE COMPILE command:

- `%LOCALE-E-CASEALRDY, case conversion already exists for '`*`character`*`'`

  Where *character* is a character from the codeset. This error can occur when the locale compiler is processing the LC_CTYPE category. It indicates that more than one case conversion is specified for *character*.

- %LOCALE-E-PREOFCMAP, premature end of file in charmap file

  Occurs if there is no END CHARMAP statement in the charmap file.

- %LOCALE-E-PREEOFSRC, premature end of file in source file

  Occurs if there is an error with the END statements in the locale source file.

- %LOCALE-F-NOADDSYM, failed to add symbol to symbol table

  Occurs when there is insufficient memory to finish the compilation. Check the amount of memory available to your process.

- %LOCALE-F-NOINITSYM, failed to initialize symbol table

  Occurs if memory is insufficient to finish the compilation. Check the amount of memory available to your process.

Examples

1.
```
$ LOCALE COMPILE EN_GB_ISO8859-1 /CHARACTER_DEFINITIONS=ISO8859-1 -
_$ /LIST /SHOW=(CHARACTER_DEFINITIONS,SYMBOLS,STATISTICS)
```

   This example shows how to generate a locale file named EN_GB_ISO8859-1.LOCALE from the source file EN_GB_ISO8859-1.LSRC, using the charmap file ISO8859-1.CMAP. To use this locale file, copy it to the SYS$I18N_LOCALE directory and set the LANG logical to "EN_GB.ISO8859-1". The listing file contains a listing of the charmap file, the symbol table, performance information, and any error messages generated by the compiler.

LOCALE LOAD

Loads the specified locale name into the system's memory as shared, read-only global data.

Format

LOCALE LOAD *locale_identifier*

Parameters

locale_identifier
Required.

Character string that identifies the locale to be loaded. Specify one of the following:

- Name of the public locale

  Specifies the public locale. Use the format:

  language_country.codeset[@modifier]

  LOCALE LOAD searches for the public locale binary file in the location defined by the logical name SYS$I18N_LOCALE. The file type defaults to .LOCALE. The period (.) and at-sign (@) characters in the name specified are replaced by underscore (_) characters.

  For example, if the name specified is "zh_CN.dechanzi@radical", LOCALE LOAD searches for the following binary locale file:

  SYS$I18N_LOCALE:ZH_CN_DECHANZI_RADICAL.LOCALE

- Name of a file

  Specifies the binary locale file. This can be any valid file specification. If either the device or directory is not specified, LOCALE LOAD first applies the current caller's device and directory as defaults. If the file is not found, the device and directory defined by the SYS$I18N_LOCALE logical name are used as defaults. The file type defaults to .LOCALE.

  Wildcards are not valid. The binary locale file cannot reside on a remote node.

Qualifiers

None.

Description

The LOCALE LOAD command loads the specified locale name into the system's memory as several shared, read-only, global sections. All processes that access the loaded locale then use this one copy of the locale, thereby reducing overall demand on system memory.

This DCL command is privileged, typically issued by the system manager. The following privileges are required:

- SYSGBL
- PRMGBL

Examples

1. `$ LOCALE LOAD JA_JP_DECKANJI`

   This example shows how to load the JA_JP_DECKANJI locale.

LOCALE UNLOAD

Unloads the specified locale name from system memory.

Format

LOCALE UNLOAD *locale*

Parameters

locale
Required.

Character string that identifies the locale to be unloaded. See the LOCALE LOAD command for acceptable formats for this parameter.

Qualifiers

None.

Description

The LOCALE UNLOAD command unloads the specified locale name from the system's memory. If a process is accessing the locale when the UNLOAD command is entered, the global sections are deleted after the process deaccesses the locale.

This DCL command is privileged, typically issued by the system manager. The following privileges are required:

- SYSGBL
- PRMGBL

Note

You can unload only locale files loaded with the LOCALE LOAD command.

Examples

1. `$ LOCALE UNLOAD JA_JP_DECKANJI`

   This example shows how to unload the JA_JP_DECKANJI locale.

LOCALE SHOW CHARACTER_DEFINITIONS

Lists character set description files (charmaps).

Format

LOCALE SHOW CHARACTER_DEFINITIONS

Parameters

None.

Qualifiers

None.

Description

The LOCALE SHOW CHARACTER_DEFINITIONS command lists the names of the character set description files (charmaps) in the public directory defined by the logical name SYS$I18N_LOCALE.

A charmap defines the symbolic names and values of characters in a coded character set. Charmaps are used by the LOCALE COMPILE command when compiling a locale. A charmap file has the file type .CMAP.

Examples

1.
```
$ LOCALE SHOW CHARACTER_DEFINITIONS

[SYS$I18N.LOCALES.SYSTEM]DECHANYU
[SYS$I18N.LOCALES.SYSTEM]DECHANZI
[SYS$I18N.LOCALES.SYSTEM]DECKANJI
[SYS$I18N.LOCALES.SYSTEM]DECKOREAN
[SYS$I18N.LOCALES.SYSTEM]EUCJP
[SYS$I18N.LOCALES.SYSTEM]EUCTW
[SYS$I18N.LOCALES.SYSTEM]ISO8859-1
[SYS$I18N.LOCALES.SYSTEM]ISO8859-2
[SYS$I18N.LOCALES.SYSTEM]ISO8859-3
[SYS$I18N.LOCALES.SYSTEM]ISO8859-4
[SYS$I18N.LOCALES.SYSTEM]ISO8859-5
[SYS$I18N.LOCALES.SYSTEM]ISO8859-7
[SYS$I18N.LOCALES.SYSTEM]ISO8859-8
[SYS$I18N.LOCALES.SYSTEM]ISO8859-9
[SYS$I18N.LOCALES.SYSTEM]MITACTELEX
[SYS$I18N.LOCALES.SYSTEM]SDECKANJI
[SYS$I18N.LOCALES.SYSTEM]SJIS
```

   This example displays the names of the charmap files, all in the SYS$I18N_LOCALE directory.

LOCALE SHOW CURRENT

Displays a summary of the current international environment as defined by several international environment logical names.

Format

LOCALE SHOW [CURRENT]

Parameters

None.

Qualifiers

None.

Description

The LOCALE SHOW CURRENT command lists the settings for each locale category and the values of the environment variables LC_ALL and LANG.

The CURRENT keyword is the default and is, therefore, optional. The logical name that defines a category has the same name as the category. For example, the LC_MESSAGES logical name defines the setting for the LC_MESSAGES category. Table 4–3 describes the locale categories.

Table 4–3 Locale Categories

Category	Description
LC_COLLATE	Information about collating sequences
LC_CTYPE	Information about character classification
LC_MESSAGES	Information about the language of program messages and the format of yes/no prompts
LC_MONETARY	Information about monetary formatting
LC_NUMERIC	Information about formatting numbers
LC_TIME	Information about time and date

Each locale category is defined by scanning the following logical names in the order shown, until a logical name is found. If the logical name found does not represent a valid locale file, LOCALE SHOW displays the string "C" for all the categories.

1. LC_ALL
2. Logical names corresponding to the categories specified in the table (For example, if LC_NUMERIC is specified as a valid locale category, the LOCALE SHOW CURRENT command displays the name of the category and the locale name it defines.)
3. LANG
4. SYS$LC_ALL

5. The system default for the locale categories as specified by the SYS$* logical names. (For example, the default for the category LC_NUMERIC is defined by the SYS$LC_NUMERIC logical name.)
6. SYS$LANG

The system manager can choose to define SYS$* logicals in the site-specific system startup files to set the default locale. If no definition is provided, programs operate using the built-in "C" locale, in which case the LOCALE SHOW CURRENT command displays the string "C" for the current locale categories.

Errors

If any logical names that define the environment are improperly defined, no warning message is issued. However, the actual international environment is listed exactly as it would be seen by an application that uses the Compaq C Run-Time Library routine `setlocale` (for instance, if in the previous example the SPECIAL.LOCALE file does not exist, then the display for the LC_MESSAGES category would show LC_MESSAGES="C").

Examples

1.
```
$ DEFINE LC_COLLATE EN_US.ISO8859-1 ! NOTE: the collate category in unquoted
$ DEFINE LANG EN_GB_ISO8859-1
$ DEFINE LC_MESSAGES PRIVATE$DISK:[APPL.LOCALES]SPECIAL.LOCALE
$ LOCALE SHOW CURRENT

LANG="EN_GB_ISO8859-1"
LC_CTYPE="EN_GB_ISO8859-1"
LC_COLLATE=EN_US_ISO8859-1
LC_TIME="EN_GB_ISO8859-1"
LC_NUMERIC="EN_GB_ISO8859-1"
LC_MONETARY="EN_GB_ISO8859-1"
LC_MESSAGES=PRIVATE$DISK:[APPL.LOCALES]SPECIAL.LOCALE;1
LC_ALL=
```

This example shows a process where all locale categories except LC_COLLATE and LC_MESSAGES have defaulted to the same locale, EN_GB.ISO8859-1. A setting enclosed in double quotation marks indicates that the setting is implied by the setting of one of the following logical names: LANG, LC_ALL, SYS$LC_ALL, or SYS$LANG. A setting not enclosed by double quotes indicates that the logical name for that category defines the international environment. This example also shows that if a locale category is specified by a complete file specification, then the complete file specification is displayed.

LOCALE SHOW PUBLIC

Lists all the public locales on the system.

Format

LOCALE SHOW PUBLIC

Parameters

None.

Qualifiers

None.

Description

The LOCALE SHOW PUBLIC command lists all the public locales on the system. The set of public locales contains all the locales that reside in the directory defined by the logical name SYS$I18N_LOCALE as well as the system's built-in locales supplied with the Compaq C Run-Time Library.

Examples

1.
```
$ LOCALE SHOW PUBLIC

C (Built-in)
POSIX (Built-in)
[SYS$I18N.LOCALES.SYSTEM]EN_GB_ISO8859_1
[SYS$I18N.LOCALES.SYSTEM]EN_US_ISO8859_1
[SYS$I18N.LOCALES.SYSTEM]FR_CA_ISO8859_1
[SYS$I18N.LOCALES.SYSTEM]GRBAGE_LOCALE (bad file header checksum)
[SYS$I18N.LOCALES.SYSTEM]JA_JP_DECKANJI (Permanently Loaded)
```

   This example shows a system with three locale files in the SYS$I18N_ LOCALE directory. The C and POSIX locales are built in with the system and, therefore, cannot be found in the SYS$I18N_LOCALE directory.

   This example also shows the effect of having a bad file or a nonlocale file in the public directory and the effect of having a locale file loaded into the system's memory by the LOCALE LOAD command.

LOCALE SHOW VALUE

Displays the value of one or more keywords from the current international environment.

Format

LOCALE SHOW VALUE *name*

Parameters

name
Required. Specifying more than one *name* is valid.

Name of one of the following:

- Keyword
  - If you specify a keyword, the value of that keyword in the current locale is displayed.
  - For integer keywords that have no value assigned, the value CHAR_MAX (127) is displayed.
  - When a keyword value includes semicolons, double quotes, backslashes, or control characters, they are preceded by an escape character (usually a backslash).
- Category

  If you specify a category, the values of the keywords in that category are displayed.

Table 4–4 lists the categories and keywords you can specify.

Table 4–4 Locale Categories and Keywords

Category	Keyword	Keyword Description
LC_CTYPE		Character classification names
LC_TIME	DAY	Full weekday names
	ABDAY	Abbreviated weekday names
	MON	Full month names
	ABMON	Abbreviated month names
	D_T_FMT	Date and time format
	D_FMT	Date format
	T_FMT	Time format
	T_FMT_AMPM	Time format in the 12-hour clock
	AM_PM	Defines how the ante meridiem (a.m.) and post meridiem (p.m.) strings are represented
	ERA	Defines how years are counted and displayed for eras in a locale

(continued on next page)

Table 4–4 (Cont.) Locale Categories and Keywords

Category	Keyword	Keyword Description
	ERA_D_FMT	Era date format
	ERA_D_T_FMT	Era date and time format
	ERA_T_FMT	Era time format
	ALT_DIGITS	String defining alternative symbols for digits
LC_NUMERIC	DECIMAL_POINT	Character used as a decimal delimiter
	THOUSANDS_SEP	Character used to group digits to the left of the decimal delimiter
	GROUPING	Defines how characters to the left of the decimal delimiter are grouped
LC_MONETARY	INT_CURR_SYMBOL	Character string representing the international currency symbol.
	CURRENCY_SYMBOL	String used as the local currency symbol.
	MON_DECIMAL_POINT	Character used as a decimal delimiter when formatting monetary quantities.
	MON_THOUSANDS_SEP	Character used as a separator for groups of digits to the left of the decimal delimiter.
	POSITIVE_SIGN	String used to represent positive monetary quantities.
	NEGATIVE_SIGN	String used to represent negative monetary quantities.
	INT_FRAC_DIGITS	Number of digits displayed to the right of the decimal delimiter when formatting monetary quantities using the international currency symbol.
	FRAC_DIGITS	Number of digits displayed to the right of the decimal delimiter when formatting monetary quantities using the local currency symbol.
	P_CS_PRECEDES	For positive monetary values, this is set to 1 if the local currency symbol precedes the number and 0 if the symbol follows the number.
	N_CS_PRECEDES	For negative monetary values, this is set to 1 if the local currency symbol precedes the number and 0 if the symbol follows the number.
	P_SEP_BY_SPACE	For positive monetary values, this is set to 0 if there is no space between the currency symbol and the value, 1 if there is a space, and 2 if there is a space between the symbol and the sign string.
	N_SEP_BY_SPACE	For negative monetary values, this is set to 0 if there is no space between the currency symbol and the value, 1 if there is a space, and 2 if there is a space between the symbol and the sign string.

(continued on next page)

Table 4–4 (Cont.) Locale Categories and Keywords

Category	Keyword	Keyword Description
	P_SIGN_POSN	Integer used to indicate where the POSITIVE_SIGN string should be placed.
	N_SIGN_POSN	Integer used to indicate where the NEGATIVE_SIGN string should be placed.
	MON_GROUPING	Defines how digits are grouped when formatting monetary values.
LC_MESSAGES	YESSTR	String representing YES in the current locale.
	NOSTR	String representing NO in the current locale.
	YESEXPR	Expression representing an affirmative response in the current locale.
	NOEXPR	Expression representing a negative response in the current locale.

Note

When an environment variable that affects the setting of the current locale points to an invalid locale, the "C" locale is set.

Other valid keywords that are not displayed by default as part of any category include:

- CHARMAP — Displays the file specification of the charmap used when the locale was created.
- CODE_SET_NAME — Defines the name of the coded character set for which the charmap file is defined.
- MB_CUR_MAX — Defines the maximum number of bytes in a multibyte character.
- MB_CUR_MIN — Defines the minimum number of bytes in a character in the coded character set.

Qualifiers

/CATEGORY
Optional. Default: No display of category name.

Displays the category name before each keyword.

/KEYWORD
Optional. Default: No display of keyword name.

Displays the keyword name before the value of a keyword.

Description

The LOCALE SHOW VALUE command displays the value of one or more keywords from the current international environment.

Errors

%LOCALE-E-NOKEYFND, no keyword *keyword-name* found

The *keyword-name* you specified is not valid. Specify only the keywords listed in Table 4–4.

Examples

1. ```
   $ LOCALE SHOW VALUE NOEXPR
   ```

   ```
   "^[nN][[:alpha:]]*"
   ```

   Issuing LOCALE SHOW VALUE without qualifiers displays the value of the NOEXPR string.

2. ```
   $ LOCALE SHOW VALUE /CATEGORY NOEXPR
   ```

   ```
   LC_MESSAGES
   "^[nN][[:alpha:]]*"
   ```

   Specifying /CATEGORY displays the category name (LC_MESSAGES) before the value of the NOEXPR string.

3. ```
   $ LOCALE SHOW VALUE /KEYWORD NOEXPR
   ```

   ```
   noexpr= "^[nN][[:alpha:]]*"
   ```

   Specifying /KEYWORD displays the keyword name before its value.

4. ```
   $ LOCALE SHOW VALUE /KEYWORD /CATEGORY NOEXPR
   ```

   ```
   LC_MESSAGES
   noexpr= "^[nN][[:alpha:]]*"
   ```

   Specifying /KEYWORD and /CATEGORY displays the category and keyword name before the keyword value.

zic

Using the data in the specified time zone source file, creates binary files containing time zone conversion information.

Format

zic [-v] ["-L" *leapseconds*] [-d *directory*] [-y *yearistype*] *infile*

Parameters

infile
Required.

Source file that zic reads.

Qualifiers

-v
Optional.

Flags if a year that appears in a data file is outside the range of years representable by time values.

"-L"
Optional.

Reads leap second information from the file with the given name. If this option is not used, no leap second information appears in the output files.

-d
Optional.

Creates time conversion information files in the named directory rather than in the standard directory.

-y
Optional.

Uses the given command file rather than yearistype when checking year types.

Description

The zic command allows the ZIC compiler to read text from the files named on the command line, and then creates the time conversion information files specified with this input.

If a file name is –, the standard input is read.

Input lines consist of fields. Any number of white space characters separate the fields. Leading and trailing white spaces on input lines are ignored. An unquoted number sign (**#**), the sharp character, in the input line introduces a comment that extends to the end of the line where this sign appears. White space characters and sharp characters can be enclosed in double quotation marks (" ") if they are to be used as part of a field. Any line that is blank after comment stripping is ignored.

Non-blank lines are expected to be one of three types:

- Rule lines (see Section 1.2.1)
- Zone lines (see Section 1.2.2)
- Link lines (see Section 1.2.3)

Examples

1. `$ zic -v  "-L" leapseco -d [-] myafrica`

   The ZIC compiler compiles the time zone source file myafrica. Based on the specified parameters, ZIC does the following:

   1. Flags years outside the representable range
   2. Builds an output file with leapsecond corrections applied
   3. Puts the result in the current directory

   For more information about date/time functions, see the *Compaq C Run-Time Library Reference Manual for OpenVMS Systems*.

Index