

INFORMIX-GLS

Programmer's Manual

Version 3.08
December 1998
Part No. 000-5196

Published by INFORMIX® Press

Informix Corporation
4100 Bohannon Drive
Menlo Park, CA 94025-1032

© 1998 Informix Corporation. All rights reserved. The following are trademarks of Informix Corporation or its affiliates:

Answers OnLine™; C-ISAM®; Cyber Planet™; Data Director™; DataBlade®; Dynamic Scalable Architecture™; Dynamic Server™; Dynamic Server™, Developer Edition™; Dynamic Server™ with Advanced Decision Support Option™; Dynamic Server™ with Extended Parallel Option™; Dynamic Server™ with MetaCube® ROLAP Option; Dynamic Server™ with Universal Data Option™; Dynamic Server™ with Web Integration Option™; Dynamic Server™, Workgroup Edition™; 4GL for ToolBus™; If you can imagine it, you can manage it™, Illustra®, INFORMIX®, Informix Data Warehouse Solutions... Turning Data Into Business Advantage™; INFORMIX®-Enterprise Gateway with DRDA®; INFORMIX®-4GL; InformixLink®, MetaCube®; NewEra™; ON-Bar™; OnLine Dynamic Server™; OnLine for NetWare®; OnLine / Secure Dynamic Server™; OpenCase®; Regency Support®; Solution Design Labs™; Solution Design Program™; SuperView®; Universal Web Connect™; ViewPoint®. The Informix logo is registered with the United States Patent and Trademark Office.

Documentation Team: Diana Chase, Beth Delson, Barbara Nomiyama, Richelle White

GOVERNMENT LICENSE RIGHTS

Software and documentation acquired by or for the US Government are provided with rights as follows: (1) if for civilian agency use, with rights as restricted by vendor's standard license, as prescribed in FAR 12.212; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce this legend.

Table of Contents

Introduction

In This Introduction	3
About This Manual	3
Types of Users	3
Software Dependencies	4
Acknowledgment	4
Assumptions About Your Locale.	4
Demonstration Databases	5
New Features	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Character-Representation Conventions	8
Additional Documentation	12
On-Line Manuals	12
HTML Documentation	12
Printed Manuals	13
Error Message Documentation	13
Documentation Notes and Release Notes.	14
Related Reading	15
Compliance with Industry Standards	15
Informix Welcomes Your Comments	16

Chapter 1

Using INFORMIX-GLS

In This Chapter	1-3
Internationalizing Programs with INFORMIX-GLS	1-3
What Is Internationalization?	1-3
What Is INFORMIX-GLS?	1-5
What Is Localization?	1-8

Using INFORMIX-GLS	1-13
Compiling and Linking INFORMIX-GLS	1-13
Initializing the INFORMIX-GLS Library	1-18
Handling INFORMIX-GLS Exceptions	1-19
Allocating Memory	1-21
Using Stream Input and Output	1-21
Running the Program	1-22
Improving Program Performance	1-22
Optimizing INFORMIX-GLS.	1-22
Processing Wide Characters	1-23

Chapter 2 Character Processing

In This Chapter	2-3
Types of Characters	2-4
Single-Byte Characters	2-5
Multibyte Characters	2-6
Character Operations	2-10
Character Classification	2-11
Case Conversion	2-15
Code-Set Conversion	2-22
String Operations	2-30
String Traversal	2-30
String Processing.	2-32
Character/String Comparison and Sorting	2-35
Other Operations	2-37
String and Character Termination	2-38
Managing Memory for Strings and Characters	2-41
Keeping Multibyte Strings Consistent	2-43

Chapter 3 Data Formatting

In This Chapter	3-3
Locale-Specific Data Formats	3-3
LC_NUMERIC Locale-File Category	3-4
LC_MONETARY Locale-File Category	3-5
LC_TIME Locale-File Category	3-6
Conversion and Formatting with INFORMIX-GLS	3-8
Converting a Locale-Specific String	3-10
Formatting a Locale-Specific String	3-11

Chapter 4

INFORMIX-GLS Function Descriptions

In This Chapter	4-5
Function Summary	4-5
Memory Allocation	4-5
Initialization and Error Handling.	4-6
Stream Input and Output	4-6
Character Processing	4-6
Data Formatting	4-9
Function Reference	4-10
ifx_gl_case_conv_outbuflen()	4-11
ifx_gl_conv_needed().	4-13
ifx_gl_convert_date().	4-15
ifx_gl_convert_datetime()	4-24
ifx_gl_convert_money()	4-35
ifx_gl_convert_number()	4-40
ifx_gl_cv_mconv().	4-45
ifx_gl_cv_outbuflen().	4-49
ifx_gl_cv_sb2sb_table()	4-52
ifx_gl_format_date()	4-54
ifx_gl_format_datetime()	4-63
ifx_gl_format_money()	4-74
ifx_gl_format_number().	4-80
ifx_gl_getmb()	4-86
ifx_gl_init()	4-89
ifx_gl_ismalnum().	4-91
ifx_gl_ismalpha().	4-93
ifx_gl_ismblank().	4-96
ifx_gl_ismcntrl().	4-98
ifx_gl_ismdigit().	4-100
ifx_gl_ismgraph().	4-102
ifx_gl_ismlower().	4-104
ifx_gl_ismprint().	4-106
ifx_gl_ismpunct().	4-108
ifx_gl_ismspace().	4-110
ifx_gl_ismupper().	4-113
ifx_gl_ismxdigit().	4-115
ifx_gl_iswalnum().	4-118
ifx_gl_iswalpha().	4-120
ifx_gl_iswblank().	4-122
ifx_gl_iswcntrl().	4-124
ifx_gl_iswdigit().	4-126
ifx_gl_iswgraph().	4-128

ifx_gl_iswlower().	4-130
ifx_gl_iswprint()	4-132
ifx_gl_iswpunct().	4-134
ifx_gl_iswspace().	4-136
ifx_gl_iswupper()	4-138
ifx_gl_iswxdigit().	4-140
ifx_gl_lc_errno()	4-142
ifx_gl_mb_loc_max()	4-144
ifx_gl_mblen().	4-145
ifx_gl_mbscat()	4-147
ifx_gl_mbschr()	4-150
ifx_gl_mbscoll()	4-152
ifx_gl_mbscpy()	4-155
ifx_gl_mbscspn()	4-157
ifx_gl_mbslen()	4-160
ifx_gl_mbsmbs()	4-162
ifx_gl_mbsncat()	4-165
ifx_gl_mbsncpy().	4-168
ifx_gl_mbsnext()	4-171
ifx_gl_mbsntsbytes()	4-173
ifx_gl_mbsntslen()	4-175
ifx_gl_mbspbrk().	4-177
ifx_gl_mbsprev()	4-180
ifx_gl_mbsrchr()	4-182
ifx_gl_mbsspn()	4-185
ifx_gl_mbstowcs()	4-188
ifx_gl_mbtowc()	4-191
ifx_gl_putmb()	4-193
ifx_gl_tomlower()	4-196
ifx_gl_tomupper()	4-199
ifx_gl_towlower()	4-202
ifx_gl_towupper()	4-204
ifx_gl_wcscat()	4-206
ifx_gl_wcschr()	4-208
ifx_gl_wcscoll()	4-210
ifx_gl_wcscpy()	4-213
ifx_gl_wscscpn()	4-215
ifx_gl_wcslen()	4-218
ifx_gl_wcsncat()	4-220
ifx_gl_wcsncpy().	4-222
ifx_gl_wcsntslen()	4-224
ifx_gl_wcspbrk()	4-226

ifx_gl_wcsrchr().	4-228
ifx_gl_wcssp().	4-230
ifx_gl_wcstombs().	4-233
ifx_gl_wcs wcs().	4-235
ifx_gl_wctomb()	4-237

Appendix A
List of INFORMIX-GLS Error Numbers

Index

Introduction

About This Manual.	3
Types of Users	3
Software Dependencies	4
Acknowledgment	4
Assumptions About Your Locale.	4
Demonstration Databases	5
New Features.	5
Documentation Conventions	5
Typographical Conventions	6
Icon Conventions	7
Comment Icons	7
Feature, Product, and Platform Icons	8
Character-Representation Conventions	8
Single-Byte Characters	9
Multibyte Characters	9
Single-Byte and Multibyte Characters in the Same String	10
White Space in Strings	10
Trailing White Spaces	11
Additional Documentation	12
On-Line Manuals	12
HTML Documentation	12
Printed Manuals	13
Error Message Documentation	13
Documentation Notes and Release Notes	14
Related Reading	15

Compliance with Industry Standards.	15
Informix Welcomes Your Comments	16

In This Introduction

This Introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

This manual describes the global language support (GLS) application programming interface (API) available in Informix products. INFORMIX-GLS provides INFORMIX-ESQL/C and DataBlade module programmers the ability to write programs (or change existing programs) to handle different languages, cultural conventions, and code sets.

Types of Users

This manual is written for DataBlade module developers and ESQL/C programmers who want to internationalize their applications with INFORMIX-GLS.

This manual assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming

If you have limited experience with relational databases, SQL, or your operating system, refer to your *Getting Started* manual for a list of supplementary titles.

Software Dependencies

This manual assumes that you are using an Informix database server that supports GLS and either INFORMIX-ESQL/C or the DataBlade API. In places where this manual presents database server-specific information, this information applies to one of the following database servers:

- Informix Dynamic Server
- Informix Dynamic Server with Advanced Decision Support and Extended Parallel Options
- Informix Dynamic Server with Universal Data Option
- Informix Dynamic Server, Workgroup Edition
- INFORMIX-SE

If you are using a database server that is not listed here, see your release notes for information about client behavior on your database server.

Acknowledgment

Portions of this manual were derived from the X/Open CAE Specifications, published by X/Open Company Ltd., U.K.

Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All culture-specific information is brought together in a single environment, called a GLS locale.

The examples in this manual are written with the assumption that you are using the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for dates, times, and currency. In addition, this locale supports the ISO8859-1 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the *Informix Guide to GLS Functionality*.

Demonstration Databases

The DB-Access utility, which is provided with your Informix database server products, includes one or more demonstration databases that contain information about a fictitious wholesale sporting-goods distributor. You can create and populate these demonstration databases with command files that are included with the database server.

Many examples in Informix manuals are based on these databases. For a complete explanation of how to create and populate the demonstration databases, refer to the *DB-Access User Manual*. For a description of the demonstration databases and their contents, see the *Informix Guide to SQL: Reference*.

The scripts that you use to install the demonstration databases reside in the **\$INFORMIXDIR/bin** directory on UNIX and in the **%INFORMIXDIR%\bin** directory on Windows.

UNIX

New Features

INFORMIX-Client Software Developer's Kit, Version 2.2, supports silent installation in UNIX environments. The installation script includes additional options that allow you to override version checking so that the installation can proceed without further user interaction.

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

The following conventions are discussed:

- Typographical conventions
- Icon conventions
- Character-representation conventions

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
<i>italics</i> italics <i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface boldface	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
monospace <i>monospace</i>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of one or more product- or platform-specific paragraphs.
→	This symbol indicates a menu item. For example, “Choose Tools→Options ” means choose the Options item from the Tools menu.



Tip: When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

Comment Icons

Comment icons identify three types of information, as the following table describes. This information always appears in italics.

Icon	Label	Description
	<i>Warning:</i>	Identifies paragraphs that contain vital instructions, cautions, or critical information
	<i>Important:</i>	Identifies paragraphs that contain significant information about the feature or operation that is being described
	<i>Tip:</i>	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Feature, Product, and Platform Icons

Feature, product, and platform icons identify paragraphs that contain feature-specific, product-specific, or platform-specific information.

Icon	Description
	Identifies information that is specific to the DataBlade API
	Identifies information that is specific to Informix Dynamic Server with Universal Data Option
	Identifies information that is specific to INFORMIX-ESQL / C
	Identifies information that is specific to UNIX platforms
	Identifies information that is specific to Windows NT, Windows 95, and Windows 98 environments

These icons can apply to an entire section or to one or more paragraphs within a section. If an icon appears next to a section heading, the information that applies to the indicated feature, product, or platform ends at the next heading at the same or higher level. A ♦ symbol indicates the end of feature-, product-, or platform-specific information that appears within one or more paragraphs within a section.

Character-Representation Conventions

Throughout this manual, examples show how single-byte and multibyte characters appear. Because multibyte characters are usually ideographic (such as Japanese or Chinese characters), this manual does not use the actual multibyte characters. Instead, it uses ASCII characters to represent both single-byte and multibyte characters. This section provides general information about how this manual represents multibyte and single-byte characters abstractly.

Single-Byte Characters

This manual represents single-byte characters as a series of lowercase letters. The format for representing one single-byte character abstractly is:

a

In this format, a stands for any single-byte character, not for the letter “a” itself.

The format for representing a string of single-byte characters is as follows:

a . . . z

In this format, a stands for the first character in the string, and z stands for the last character in the string. For example, if the string `Ludwig` consists of single-byte characters, the following format represents this 6-character string abstractly:

abcdef



Tip: The letter “s” does not appear in alphabetical sequences that represent strings of single-byte characters. The manual reserves the letter “s” as a symbol that represents a single-byte white-space character. For further information, see [“White Space in Strings” on page 10](#).

Multibyte Characters

This manual does not attempt to show the actual appearance of multibyte characters in text, examples, or diagrams. Instead, the following convention shows abstractly how multibyte characters are stored:

A¹ . . . Aⁿ

One to four identical uppercase letters, each followed by a different superscript number, represent one multibyte character. The superscripts show the first to the *n*th byte of the multibyte character, where *n* has values between two and four. For example, the following symbols represent a multibyte character that consists of two bytes:

A¹A²

The following notation represents a multibyte character that consists of four bytes (the maximum length of a multibyte character):

A¹A²A³A⁴

The following example shows a string of multibyte characters in an SQL statement:

```
CREATE DATABASE A1A2B1B2C1C2D1D2E1E2;
```

This statement creates a database whose name consists of five multibyte characters, each of which is two bytes long. For more information on how to use multibyte characters in SQL identifiers, see the *Informix Guide to GLS Functionality*.

Single-Byte and Multibyte Characters in the Same String

If you are using a multibyte code set, a given string might be composed of both single-byte and multibyte characters. To represent these mixed strings, this manual simply combines the formats for multibyte and single-byte characters.

For example, suppose that you have a string with four characters. The first and fourth characters are single-byte characters, and the second and third characters are multibyte characters that consist of two bytes each. The following format represents this string:

```
aA1A2B1B2b
```

White Space in Strings

White space is a series of one or more space characters. A GLS locale defines what characters are considered to be space characters. For example, both the TAB and blank might be defined as space characters in one locale, but certain combinations of the CTRL key and another character might be defined as space characters in a different locale.

The convention for representing single-byte white spaces in this manual is the letter “s.” The following notation represents one single-byte white space:

```
s
```

In the ASCII code set, an example of a single-byte white space is the blank character (ASCII code number 32). To represent a string that consists of two ASCII blank characters, the manual uses the following notation:

```
s s
```

The following notation represents a multibyte white-space character:

$$s^1 \dots s^n$$

In this format, s^1 represents the first byte of the white-space character, while s^n represents the last byte of the white-space character, where n has values between 2 and 4. For example, the following notation represents one 4-byte white-space character:

$$s^1 s^2 s^3 s^4$$

Trailing White Spaces

Combinations of characters and white spaces can occur in quoted strings, in CHAR columns that contain fewer characters than the defined length of the column, and in other situations. For example, if a CHAR(5) column in a single-byte code set contains a string of three characters, the string is extended with two white spaces so that its length is equal to the defined length of the column, as follows:

abc ss

The following example shows the representation for a string of five characters (three characters of data and two trailing white spaces) in a multibyte code set where each of the characters and white-space characters consists of two bytes:

$$A^1 A^2 B^1 B^2 C^1 C^2 s^1 s^2 s^1 s^2$$

Sometimes a string can contain both single-byte and multibyte white-space characters. In the following example, the string is composed of these elements: three single-byte characters (abc), a single-byte white-space character (s), a multibyte white-space character ($s^1 s^2$), two single-byte white-space characters (ss), and one multibyte white-space character ($s^1 s^2$):

$$abc s s^1 s^2 s s s^1 s^2$$

Additional Documentation

For additional information, you might want to refer to the following types of documentation:

- On-line manuals
- Printed manuals
- HTML documentation
- Error message documentation
- Documentation Notes, Release Notes, Machine Notes
- Related reading

On-Line Manuals

An Answers OnLine CD that contains Informix manuals in electronic format is provided with your Informix products. You can install the documentation or access it directly from the CD. For information about how to install, read, and print on-line manuals, see the installation insert that accompanies Answers OnLine.

HTML Documentation

The INFORMIX-GLS manual pages are available as HTML documentation. You can access these manual pages with a Web browser and the appropriate pathname.

If you are using INFORMIX-GLS on a UNIX operating system, the pathname is **\$INFORMIXDIR/doc/gls_api/en_us/0333/index.htm**. ♦

If you are using INFORMIX-GLS in a Windows environment, the pathname is **%INFORMIXDIR%\doc\gl_s_api\en_us\04e4\index.htm**. ♦

For example, if **INFORMIXDIR** is set to **/u/informix** on a UNIX operating system, you would type the following absolute pathname to open the file:

```
file:/u/informix/doc/gls_api/en_us/0333/index.htm
```

UNIX

Windows

Printed Manuals

To order printed manuals, call 1-800-331-1763 or send email to moreinfo@informix.com. Please provide the following information when you place your order:

- The documentation that you need
- The quantity that you need
- Your name, address, and telephone number

Error Message Documentation

Informix software products provide ASCII files that contain all of the Informix error messages and their corrective actions.

To read error messages and corrective actions on UNIX, use one of the following utilities.

Utility	Description
finderr	Displays error messages on line
rofferr	Formats error messages for printing

◆

UNIX

Windows

To read error messages and corrective actions in Windows environments, use the **Informix Find Error** utility. To display this utility, choose **Start→Programs→Informix** from the Task Bar. ◆

Instructions for using the preceding utilities are available in Answers OnLine. Answers OnLine also provides a listing of error messages and corrective actions in HTML format.

UNIX

Documentation Notes and Release Notes

In addition to printed documentation, the following sections describe the on-line files that supplement the information in this manual. Please examine these files before you begin using your database server and client products. They contain vital information about application and performance issues.

On UNIX platforms, the following on-line files appear in the **\$INFORMIXDIR/release/en_us/0333** directory.

On-Line File	Purpose
GLSAPIDOC_3.08	The documentation-notes file for your version of this manual describes features that are not covered in the manual or that were modified since publication.
CLIENTS_2.2	The release-notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds. The release-notes file for Client SDK includes information about database server compatibility.



Windows

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix** from the Task Bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals, along with information about features that might not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds. The release-notes file for Client SDK includes information about database server compatibility.



Related Reading

The following publications provide additional information about the topics that this manual discusses. For a list of publications that provide an introduction to database servers and operating-system platforms, refer to your *Getting Started* manual.

For additional information about cultural and linguistic requirements around the world, and how computer systems handle those requirements, consult the following books on internationalization:

- *Developing and Localizing International Software* by Tom Madell, Clark Parsons, and John Abegg (Prentice-Hall, Inc., 1994)
- *Programming for the World: A Guide to Internationalization* by Sandra Martin O'Donnell (Prentice-Hall, Inc., 1994)

Compliance with Industry Standards

The GLS feature for Informix products is based on the X/Open XPG4 specifications. The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

Informix Welcomes Your Comments

Let us know what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
SCT Technical Publications Department
4100 Bohannon Drive
Menlo Park, CA 94025

If you prefer to send electronic mail, our address is:

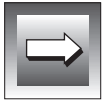
`doc@informix.com`

The **doc** alias is reserved exclusively for reporting errors and omissions in our documentation.

We appreciate your suggestions.

Using INFORMIX-GLS

Internationalizing Programs with INFORMIX-GLS	1-3
What Is Internationalization?	1-3
What Is INFORMIX-GLS?	1-5
Internationalization and INFORMIX-GLS	1-5
INFORMIX-GLS Compatibility	1-6
What Is Localization?	1-8
Choosing a GLS Locale	1-9
Translating Messages	1-12
Using INFORMIX-GLS	1-13
Compiling and Linking INFORMIX-GLS	1-13
INFORMIX-GLS in ESQL / C Applications	1-14
INFORMIX-GLS in DataBlade Programs	1-14
Initializing the INFORMIX-GLS Library	1-18
Handling INFORMIX-GLS Exceptions	1-19
Allocating Memory	1-21
Using Stream Input and Output	1-21
Running the Program	1-22
Improving Program Performance	1-22
Optimizing INFORMIX-GLS	1-22
Processing Wide Characters	1-23



In This Chapter

This chapter introduces INFORMIX-GLS. It provides information on the following topics:

- How the INFORMIX-GLS library can help you internationalize programs
- How to use the INFORMIX-GLS library
- How to optimize programs that use the INFORMIX-GLS library

Important: *The INFORMIX-GLS library enables a program to access the culture-specific information in the GLS locales that Informix provides. For an introduction to GLS concepts and use, refer to the “Informix Guide to GLS Functionality.”*

Internationalizing Programs with INFORMIX-GLS

This section provides the following information about to how to internationalize database applications with INFORMIX-GLS:

- A summary of the internationalization process
- A description of the INFORMIX-GLS library
- A summary of the localization process for a database application

What Is Internationalization?

Internationalization is the process of creating (or modifying) an application so that you point the application to the correct GLS locale to support different languages, territories, and code sets without changing or recompiling the code. This process makes Informix database applications easily adaptable to any culture and language.



For a database application, you perform internationalization on the application that accesses a database, not on the database. The data in a database that the application accesses should already be in a language that the end user can understand.

Tip: The term “internationalization” is sometimes abbreviated as “I18n,” which represents the first and last letters of the word and the 18 letters in between.

To internationalize a database application, design the application so that the tasks in [Figure 1-1](#) do not make any assumptions about the language, territory, and code set that the application uses at runtime.

Figure 1-1
Internationalization Tasks for a Database Application

Application Task	Description
User interfaces	Includes any text that is visible to end users, including menus, buttons, prompts, help text, status messages, error messages, and graphics.
Character processing	Includes the following processing tasks: ■ Character classification ■ Character case conversion ■ Collation and sorting ■ Character versus byte processing ■ String traversal ■ Code-set conversion
Data formatting	Includes any culture-specific formats for the following types of data: ■ Numeric ■ Monetary ■ Date ■ Time
Documentation	Includes any explanatory material such as printed manuals, on-line documentation, and README files.

An internationalized application dynamically obtains language-specific information for these application tasks. Therefore, one executable file for the application can support multiple languages.

What Is INFORMIX-GLS?

INFORMIX-GLS is an application programming interface (API) that lets DataBlade module developers and ESQL/C programmers develop internationalized applications with a C-language interface. You can use the library of C language functions and macros of INFORMIX-GLS to write programs (or change existing programs) to handle different languages, cultural conventions, and code sets.

Internationalization and INFORMIX-GLS

Of the internationalization tasks in [Figure 1-1 on page 1-4](#), the INFORMIX-GLS library facilitates the following ones:

- Character processing

The INFORMIX-GLS library provides functions that perform the following character-processing tasks:

- ❑ Process single-byte, multibyte, and wide characters and strings
- ❑ Collate single-byte and multibyte characters and strings
- ❑ Process input and output
- ❑ Convert between compatible code sets

For a more detailed discussion of the character processing that the INFORMIX-GLS library provides, see [Chapter 2, “Character Processing.”](#)

- Data formatting

The INFORMIX-GLS library also provides functions that convert date, time, monetary, and numeric strings from and to internal representations. For a more detailed discussion of the data formatting that the INFORMIX-GLS library provides, see [Chapter 3, “Data Formatting.”](#)

Of the internationalization tasks in [Figure 1-1 on page 1-4](#), the INFORMIX-GLS library does *not* provide functions for the following ones:

- User interfaces

You must ensure that user interfaces are designed in a way that is not language specific. In addition, put language-specific text, menus, buttons, and messages in external files that the application can reference at runtime.

- Documentation

You must ensure that any documentation you provide is translated to the appropriate language for a particular locale and territory. You might want to develop rules for the creation of documentation that simplify the translation tasks.



Important: Although the INFORMIX-GLS library does provide functions to perform case conversions (from lowercase to uppercase and vice versa), it does not provide a way to translate words, nor does it translate data in a database. You must obtain accurate and appropriate translations of text that the end user sees. These tasks must be performed by people experienced in the application and the languages involved.

INFORMIX-GLS Compatibility

The INFORMIX-GLS library is compatible with most Informix products. It also conforms to the X/Open XPG4 internationalization standards. The following sections provide additional information about INFORMIX-GLS compatibility.

With Informix Products

In order for you to use INFORMIX-GLS for internationalization, both the database server and the application must support the following Informix application products:

E/C

- INFORMIX-ESQL/C develops client applications in the C language and can contain embedded SQL statements.

For more information on how to write ESQL/C applications, see the *INFORMIX-ESQL/C Programmer's Manual*. ♦

DB API

- The Informix DataBlade API supports development of:
 - client applications.
 - user-defined routines (UDRs). ♦

Both kinds of DataBlade programs use DataBlade API functions to communicate with Informix Dynamic Server with Universal Data Option. For more information on how to use the DataBlade API, see the *DataBlade API Programmer's Manual*. ♦

IDS/UD

An ESQL/C or DataBlade API client application that uses INFORMIX-GLS can obtain locale information from an Informix database server that supports GLS functionality:

- Informix Dynamic Server
- Informix Dynamic Server with Advanced Decision Support and Extended Parallel Options
- Informix Dynamic Server with Universal Data Option
- Informix Dynamic Server, Workgroup Edition
- INFORMIX-SE
- INFORMIX-OnLine Dynamic Server
- INFORMIX-OnLine Workgroup Server

IDS/UD

A DataBlade server routine can use INFORMIX-GLS to obtain locale information from Dynamic Server with UD Option. ♦

With X/Open XPG4 Internationalization Standards

INFORMIX-GLS functions are derived from the X/Open XPG4 interface, but INFORMIX-GLS does *not* rely on the existence or absence of this interface (or any of the interfaces from which XPG4 is derived) on a particular operating system.

The name of each INFORMIX-GLS data type, macro, and function is intentionally different than the X/Open name from which it is derived. Where it benefits Informix products, INFORMIX-GLS extends or changes the X/Open functions. Therefore, the operating-system documentation for the internationalization functions does not adequately describe how to use the INFORMIX-GLS functions.

What Is Localization?

Localization is the process of adapting a product to a specific cultural environment. This process usually involves the following tasks:

- Creating culture-specific resource files
- Translating message or resource files
- Setting date, time, and money formats
- Translating the product user interface

Localization might also include the translation and production of end-user documentation, packaging, and collateral materials.

Tip: The term “localization” is sometimes abbreviated as “L10n,” which represents the first and last letters of the word and the 10 letters in between.



To localize a database application, you create a database application for a specific language, territory, and code set. Localization involves the following tasks:

- Ensure that GLS locales exist for the desired language, territory, and code set.
- Translate the character strings in any external resource or message files that the application uses.

Important: *An internationalized application is much easier to localize than a non-internationalized application.*

Choosing a GLS Locale

An Informix *GLS locale* is a set of Informix files that bring together information about data that is specific to a particular culture, language, or territory. In particular, a GLS locale provides the following information:

- The name of the code set that the application data uses
- The classes of characters in the code set
- The collation order to use for character data
- The format for date, time, numeric, and monetary data to appear to end users

To localize your application, choose a locale that provides the culture-specific information for the language, territory, and code set that the application is to support. Locale files are located in various subdirectories under the **gls** directory of the **INFORMIXDIR** directory.

Important: *Informix defines and preprocesses these locale files. You cannot modify Informix locales.*



At runtime, a database application uses the following locales.

Locale	Definition	Purpose
Client locale	The locale that the database application uses for locale-specific information	The application uses the client locale to obtain: ■ the code set for read and write (I/O) operations on the client computer. ■ the format for literal data strings (data formats). ■ the code set for embedded SQL statements, host variables, and data sent to or received from the database server.
Database locale	The locale that the database uses for its data	The application uses the database locale to obtain the code set for: ■ data sent to or received from the database server. ■ names of database objects, such as databases, tables, columns, and views. If this code set is different from the client code set, the application <i>must</i> perform code-set conversion.
Server locale	The locale that the database server uses for locale-specific information	When the client application establishes a connection, the database server uses the client locale, database locale, and server locale to determine the <i>server-processing locale</i> for this connection. The database server uses the server-processing locale for its own internal sessions and for access to any database that the connection opens.

At runtime, an application obtains the locales that it needs from its application environment. You establish the locales of the application environment as follows.

Locale	Default Locale	Nondefault Locale
Client locale	U.S. English	Set the CLIENT_LOCALE environment variable to the name of the desired locale.
Database locale	U.S. English	Set the DB_LOCALE environment variable to the name of the desired locale.
Server locale	U.S. English	Set the SERVER_LOCALE environment variable to the name of the desired locale.

At runtime, an internationalized application makes no assumptions about how these locales are set. Once the application environment specifies the locales to use, the application can access the appropriate GLS locale files for locale-specific information. As long as Informix provides a GLS locale that supports a particular language, territory, and code set, the application can obtain the locale-specific information dynamically.

The *current processing locale* (sometimes called the *current locale*) is the locale currently in effect for an application. It is based on one of the following environments:

- The client environment
ESQL/C creates client applications. Therefore, the current processing locale for ESQL/C applications is the client locale. ♦
The current processing locale for DataBlade client applications is the client locale. ♦
- The database that the database server is currently accessing
The current processing locale for DataBlade UDRs is the server-processing locale, which the database server determines from the client, database, and server locales. ♦

Tip: For more information about the default locale, client locale, database locale, server locale, server-processing locale, or any of the locale environment variables, see the “Informix Guide to GLS Functionality.”



E/C

DB API

DB API

IDS/UD

Translating Messages

An internationalized application should not have any language-specific text within the application code. This language-specific text includes the following kinds of strings:

- Strings that the application displays or writes
Examples include error messages, informational messages, menu items, and button labels.
- Strings that the application uses internally
Examples include constants, filenames, and literal characters or strings.
- Strings that an end user is expected to enter
Examples include yes / no responses.

Tip: *You do not need to put SQL keywords (such as **SELECT**, **WHERE**, **INSERT**, and **CREATE**) in a message file. In addition, language keywords (such as **if**, **switch**, **for**, and **char**) do not need to appear in this message file.*

In an internationalized application, these strings appear as references to external files, called *resource files* or *message files*. To localize these strings of the database application, you must:

- translate all strings within the external files.
The new external files contain the translated versions of the strings that the application uses.
- set the **DBLANG** environment variable to the subdirectory within **INFORMIXDIR** that contains the translated message files that the Informix products use.
The **INFORMIXDIR** environment variable indicates the directory in which the Informix products are installed. You can use the **rgetmsg()** and **rgetlmsg()** functions to obtain messages for Informix products. For more information on these functions, see the *INFORMIX-ESQL/C Programmer's Manual*.

Tip: *For more information about how to use the **DBLANG** environment variable, see the “Informix Guide to GLS Functionality.”*



Using INFORMIX-GLS

To use INFORMIX-GLS in a C-language program, include the following header file in the source file:

```
#include <ifxgls.h>
```

“[Compiling and Linking INFORMIX-GLS](#)” on page 1-13 lists the directory in which this header file is located.

This section provides the following additional information about how to use the INFORMIX-GLS library:

- Compiling and linking the INFORMIX-GLS library with your application
- Initializing the server-processing locale for the INFORMIX-GLS library
- Handling exceptions from the INFORMIX-GLS functions
- Allocating memory for structures and strings
- Using stream input and output (I/O)

For more information about a particular INFORMIX-GLS function, see [Chapter 4, “INFORMIX-GLS Function Descriptions.”](#)

Compiling and Linking INFORMIX-GLS

The following table lists the directories that must be accessible to compile and link an ESQL/C or DataBlade program with INFORMIX-GLS.

Contents	Windows Directory	UNIX Directory
Subdirectories for GLS locale and code-set conversion files	%INFORMIXDIR%\gls	\$INFORMIXDIR/gls
Static and shared GLS libraries	%INFORMIXDIR%\lib\esql	\$INFORMIXDIR/lib/esql
Two GLS header files: gls.h and ifxgls.h	%INFORMIXDIR%\incl\public	\$INFORMIXDIR/incl/public

E/C

In all these directories, the **INFORMIXDIR** environment variable is set to the directory in which your Informix products are installed.

INFORMIX-GLS in ESQL/C Applications

To compile and link ESQL/C applications that use INFORMIX-GLS, issue the following command:

```
% esql source_file
```

The **esql** preprocessor automatically links the INFORMIX-GLS library to an ESQL/C application. It also links the version of the GLS libraries (shared or static) that you specify with command-line options. For more information on how to compile an ESQL/C application, see the *INFORMIX-ESQL/C Programmer's Manual*. ♦

DB API

INFORMIX-GLS in DataBlade Programs

The way to compile and link a DataBlade program depends on whether the program is:

- a DataBlade client application.
- a DataBlade user-defined routine (UDR). ♦

IDS/UD

DataBlade Client Applications

This section provides information for how to compile and link a DataBlade client application with the INFORMIX-GLS libraries. For information on how to write and compile DataBlade client applications, see the *DataBlade API Programmer's Manual*.

You must specify the location of the INFORMIX-GLS header file, **ifxgls.h**. This header file is located in **%INFORMIXDIR%\incl\public** for Windows and in **\$INFORMIXDIR/incl/public** for UNIX.

This directory also contains many of the other files that a DataBlade client application uses. Therefore, it is an important directory to include when you compile the application.

Windows

The following command uses the Microsoft compiler, **cl**, to compile a DataBlade client application called **sample.c** that uses INFORMIX-GLS:

```
cl -MD -Id:\msdev\include -Id:\informix\incl\public -Id:\informix\incl\esql \
-Id:\informix\incl -c sample.c
```

All INFORMIX-GLS functions are exported through the **libthdmi.lib** library. For linking, you must specify the following two libraries.

At Link Time (in lib Subdirectory of INFORMIXDIR)	At Runtime (in bin Subdirectory of INFORMIXDIR)	Purpose
libthdmi.lib	idmit09a.dll	DataBlade API functions
isqlt09a.lib	isqlt09a.dll	ESQL/C functions

UNIX

Make sure to include the **\$INFORMIXDIR/incl/public** directory with the **-I** compiler option, as follows:

```
cc -I$INFORMIXDIR/incl/public -I$INFORMIXDIR/esql -I$INFORMIXDIR/incl -c sample.c
```

The preceding command checks to see whether the API client libraries have been installed in the directory that the **INFORMIXDIR** environment variable indicates.

On UNIX platforms, you must explicitly specify the location and name of the library, as follows:

- For the library location, use the following **-L** compiler option:
-L\$INFORMIXDIR/lib/esql
- For the library name, include the **-lixgls** command-line option.

The following command links the **sample.o** object file to create an executable file called **sample**:

```
cc -o sample sample.o \
-L$INFORMIXDIR/lib/esql -L$INFORMIXDIR/lib/dmi -L$INFORMIXDIR/lib \
-lifdmi -lifsql -lifasf -lifcss -lifos -lifgen -lifgls -lifglx \
-lns1 -lsocket -laio -lm -ldl -lelf -lw \
$INFORMIXDIR/lib/esql/checkapi.o
```

Alternatively, you can use the ESQL/C preprocessor, **esql**, to compile and link the DataBlade client application. The **esql** preprocessor automatically links most of the libraries that the client application requires (including the INFORMIX-GLS library). However, you must explicitly specify the **libdmi** library. The following **esql** command compiles and links the **sample.o** object file:

```
esql -o sample sample.o -L$INFORMIXDIR/lib/dmi -ldmi
```



DataBlade UDRs

This section provides information about how to compile and link a DataBlade UDR. For more information about how to write user-defined routines, see *Extending Informix Dynamic Server*, the *DataBlade API Programmer's Manual*, and the DataBlade Developers Kit documentation.

The GLS libraries are already linked to the database server. All INFORMIX-GLS functions are exported through the SAPI.LIB library. Therefore, you do not need to specify the INFORMIX-GLS library explicitly when you compile and link a DataBlade UDR on Windows.

The following sample commands compile and link a DataBlade UDR called **func1()** that uses the INFORMIX-GLS library:

```
c1 /DNT_MI_SAPI /DMI_SERVBUILD
  -Id:\msdev\include -Id:\informix\incl\public -Id:\informix\incl
  -c func1.c
link /DLL /OUT:func1.dll /DEF:func1.def func1.obj d:\informix\lib\SAPI.LIB
```

In the preceding example, **d:** specifies the drive on which Informix software is installed; **informix** is the directory that **INFORMIXDIR** specifies. The resulting dynamic link libraries (DLLs) must have the READONLY attribute set to be executable. Set the READONLY flag with the **attrib +r** command.

The Microsoft compiler, **cl**, defaults to the **/MT** option, which specifies the multithreaded library for DLL builds. This multithreaded library is statically linked to the DLL. You can specify the **/MD** option to put **MSVCRT40.DLL** in a command path. All the UDRs that are built with **/MD** use one copy of the LIBC DLL. This option prevents virtual-memory buildup in the database server when many UDRs are loaded.

DB API

IDS/UD

Windows

The following sample command line shows the use of the **/MD** option to compile the **func1Q** DataBlade UDR:

```
c1 /MD /DNT_MI_SAPI /DMI_SERVBUILD \
    -Id:\msdev\include -Id:\informix\incl\public -Id:\informix\incl
    -c func1.c
```

If **MSVCRT40.DLL** is not visible to the database server, the database server puts the following error in the **online.log** file when it attempts to execute the UDR:

```
Error loading UDR
```



UNIX

On UNIX platforms, the GLS libraries are *not* linked to the database server. Therefore, you must explicitly specify the location and name of the INFORMIX-GLS library, as you do for DataBlade client applications. For more information, see [“DataBlade Client Applications” on page 1-14](#).

The location of the INFORMIX-GLS header file, **ifxgls.h**, is the following directory:

```
$INFORMIXDIR/incl/public
```

This directory contains many of the other files that a DataBlade client application uses. Make sure that you include this directory with the **-I** option of the compiler.



Tip: When you use the DataBlade Developers Kit to compile user-defined routines, you do not have to specify the location of the header files explicitly.

Additionally, you need to indicate that the DataBlade module is a DataBlade UDR (which runs on the server computer), not a DataBlade client application (which runs on a client computer). Include the following compiler flag when you compile a DataBlade UDR:

```
MI_SERVBUILD
```

The following sample compilation and link builds a shared object called **udrs.so** that contains the C code for a DataBlade UDR called **func1Q**:

```
% cc -KPIC -DMI_SERVBUILD -I$INFORMIXDIR/incl/public -I$INFORMIXDIR/incl
    -L$INFORMIXDIR/esql/lib -c func1.c
% ld -G func1.o -o udrs.so
```



Initializing the INFORMIX-GLS Library

Before you can use INFORMIX-GLS functions in your application, these functions must be able to access the current processing locale. The **ifx_gl_init()** function initializes the current processing locale to the client locale. It creates a global locale structure that the INFORMIX-GLS functions can access to obtain the name of the current processing locale.

Whether an application requires a call to the **ifx_gl_init()** function depends on whether the application establishes its own connection to a database server or executes in the context of an existing database server connection:

- ESQL/C applications establish their own connections to a database server and therefore must call the **ifx_gl_init()** function at the beginning of the **main()** program block to establish the client locale as the current processing locale. ♦
- DataBlade client applications establish their own connections to a database server and therefore must call the **ifx_gl_init()** function at the beginning of the **main()** program block to establish the client locale as the current processing locale.
- DataBlade UDRs execute in the context of an established connection and therefore use the server-processing locale as their current processing locale.

The database server uses a server-processing locale to obtain locale information for its own internal sessions and for any connections. It uses the values of the client locale, database locale, and server locale, as well as other environment information to determine the server-processing locale. DataBlade UDRs are *not* required to call the **ifx_gl_init()** function. For more information on the server-processing locale and how the database server establishes it, see the *Informix Guide to GLS Functionality*. ♦

Important: You must ensure that the database server has established a current processing locale for an application before the application uses any other INFORMIX-GLS functions. If an application requires the **ifx_gl_init()** function, it must call **ifx_gl_init()** before it calls any other INFORMIX-GLS functions.

The **ifx_gl_init()** function does not generate an error if you call it multiple times within the same application. The first time that the application calls **ifx_gl_init()**, the function initializes the global locale structure. Subsequent calls do not result in multiple locale structures.

E/C

DB API

IDS/UD



Handling INFORMIX-GLS Exceptions

Many of the INFORMIX-GLS functions return a unique value if they encounter an error. To further identify the cause of the error, you can use the **ifx_gl_lc_errno()** function to obtain the value of the INFORMIX-GLS *error number*. The **ifx_gl_lc_errno()** function returns the error number as an **int** value. INFORMIX-GLS library functions can use **ifx_gl_lc_errno()** to obtain more information about an error that has occurred. The INFORMIX-GLS functions set the error number only if an error occurs unless a particular function is documented otherwise.



Tip: For an alphabetical list of valid error-number values and the INFORMIX-GLS functions that might set them, see [Appendix A, “List of INFORMIX-GLS Error Numbers.”](#)

Because any INFORMIX-GLS library function can set the error number, you must call **ifx_gl_lc_errno()** *immediately* after the INFORMIX-GLS function in which the error occurred to inspect the relevant value of the error number.

For example, the **ifx_gl_mbslen()** function returns a value of -1 when it encounters an error. The following code fragment shows how to use the **ifx_gl_lc_errno()** function to obtain the error number:

```
if ( ifx_gl_mbslen(mbs, n) == -1 )
    switch ( ifx_gl_lc_errno() )
    ...
```

An application can also ignore the return value of an INFORMIX-GLS function and just check **ifx_gl_lc_errno()** as long as the application sets **ifx_gl_lc_errno()** to 0 *before* it calls the function.

```
ifx_gl_lc_errno() = 0;
(void) ifx_gl_mbslen(mbs, n);
if ( ifx_gl_lc_errno() != 0 )
    switch ( ifx_gl_lc_errno() )
    ...
```

If an INFORMIX-GLS library function does *not* return a unique value to indicate that an error has occurred, you must use the preceding technique to obtain the error number. For example, the following code fragment tests whether the **ifx_gl_ismupper()** function has detected an invalid multibyte character:

```
ifx_gl_lc_errno() = 0;
value = gl_ismupper(mb, IFX_GL_NO_LIMIT);
if ( ifx_gl_lc_errno() != 0 )
    switch ( ifx_gl_lc_errno() )
    ...
```

You can set **ifx_gl_lc_errno()** to 0 and check it after a function call *only* if the current processing locale was initialized correctly with **ifx_gl_init()**. For example, the following code fragment of a client application results in a memory fault because the current processing locale is uninitialized:

```
int wrong_func()
{
    ifx_gl_lc_errno() = 0;    /* A memory fault will occur here. */
    (void) ifx_gl_init();
    if ( ifx_gl_lc_errno() != 0 )
        switch ( ifx_gl_lc_errno() )
        ...
```

In the preceding code fragment, the **ifx_gl_lc_errno()** function can write or read the error number only after a call to the **ifx_gl_init()** function has correctly initialized the current processing locale.

The preceding code fragment would work in a DataBlade UDR because a DataBlade UDR does not need to call **ifx_gl_init()** to initialize the current processing locale. ♦

The following code fragment of a client application executes successfully because it correctly initializes the server-processing locale:

```
int right_func()
{
    (void) ifx_gl_init();
    ifx_gl_lc_errno() = 0;
    if ( ifx_gl_lc_errno() != 0 )
        switch ( ifx_gl_lc_errno() )
        ...
```

DB API

IDS/UD

E/C

For multithreaded ESQL/C applications, the error-number value is stored as a field in the thread control block. Therefore, multithreaded ESQL/C applications can use the error number. ♦

Allocating Memory

If an INFORMIX-GLS function allocates memory, this memory is only allocated within the function itself and is freed before the function returns. No INFORMIX-GLS library function allocates memory that remains after the function returns. Therefore, you must allocate any memory for data that an INFORMIX-GLS function needs or returns.

DB API

IDS/UD

DataBlade UDRs can assume that the INFORMIX-GLS functions allocate memory with a PER_ROUTINE memory duration. ♦

You must ensure that memory is allocated for any return value or argument that an INFORMIX-GLS function provides to the calling program.



Tip: You must also handle memory allocation for multibyte-character and wide-character strings that an INFORMIX-GLS functions use. For more information, see [“Managing Memory for Strings and Characters” on page 2-41](#).

Using Stream Input and Output

Character data that contains Asian characters must be correctly processed in all I/O for a graphical user interface, clipboard, character terminal, file, and network. The following INFORMIX-GLS functions process input and output multibyte character streams:

- The **ifx_gl_getmb()** function calls a function that you define to obtain the bytes that form one multibyte character from a specified location.
- The **ifx_gl_putmb()** function calls a function that you define to put the bytes that form one multibyte character in a specified location.

Running the Program

A client application (ESQL/C or DataBlade) that uses the INFORMIX-GLS library must be able to access the following directories at runtime.

Contents	Windows Directory	UNIX Directory
Subdirectories for locale and code-set conversion files	%INFORMIXDIR%\gls	\$INFORMIXDIR/gls
Static and shared GLS libraries	%INFORMIXDIR%\lib\esql	\$INFORMIXDIR/lib/esql

DB API

IDS/UD

A DataBlade UDR executes in the context of the SQL statement that called it. The calling program must have access to the preceding directories. ♦

Improving Program Performance

Performance of an application that uses INFORMIX-GLS is enhanced in the following ways:

- Optimization that all INFORMIX-GLS functions automatically perform
- Optional use of wide characters in the application

Optimizing INFORMIX-GLS

You can optimize the INFORMIX-GLS functions to perform the following tasks:

1. Evaluate the requested INFORMIX-GLS function.
2. Determine if the requested function is appropriate for the data that you are trying to process.
3. Execute the code that is appropriate for the type of data.

For example, if you use the **ifx_gl_mbsnext()** function to traverse data that is encoded in a single-byte code set, the function is reduced to a macro that advances the character byte by byte. The function does *not* execute code that parses multibyte sequences because it determines that the data is single byte.

Additionally, when collation is based on code-set order rather than localized order, functions use a binary compare (such as **strcmp()**) instead of algorithms that examine collation weights.

Processing Wide Characters

You can choose how to structure your data to improve performance. INFORMIX-GLS provides wide-character versions of most of its multibyte processing functions. Typically, wide-character processing functions are faster than multibyte-character functions. However, wide characters require more space. Therefore, character data is generally stored and retrieved from the database in its multibyte form.

To use wide-character processing

1. Convert the multibyte string to a wide-character string with the **ifx_gl_mbstowcs()** function.
2. Process the wide-character string.
3. Convert the wide-character string back to a multibyte string with the **ifx_gl_wcstombs()** function.

This technique is cost effective if the data that you process is traversed more than once. For more information on wide characters, see [“Wide Characters” on page 2-8](#).

Character Processing

Types of Characters	2-4
Single-Byte Characters	2-5
Multibyte Characters	2-6
The <code>gl_mchar_t</code> Data Type	2-7
Wide Characters	2-8
Character Operations	2-10
Character Classification	2-11
Case Conversion	2-15
Case Conversion for Multibyte Characters	2-16
Case Conversion for Wide Characters	2-20
Exception Handling	2-21
Performance Issues	2-21
Code-Set Conversion	2-22
Determining If Code-Set Conversion Is Needed	2-22
Performing Code-Set Conversion	2-23
Performance Issues	2-29
String Operations	2-30
String Traversal	2-30
Multibyte-Character-String Traversal	2-30
Wide-Character-String Traversal	2-31
String Processing	2-32
Concatenation	2-32
String Copying	2-33
String-Length Determination	2-33
Character Searching	2-34
Character / String Comparison and Sorting	2-35

Other Operations	2-37
String and Character Termination	2-38
Character-String Termination.	2-38
Multibyte-Character Termination	2-39
Managing Memory for Strings and Characters	2-41
Multibyte-Character-String Allocation	2-41
Wide-Character-String Allocation	2-42
String Deallocation	2-43
Keeping Multibyte Strings Consistent	2-43
Truncating Multibyte Strings	2-44
Fragmenting Multibyte Strings	2-44

In This Chapter

This chapter describes the character-processing capabilities of the INFORMIX-GLS library. It covers the following topics that the INFORMIX-GLS library supports:

- Character types: single byte, multibyte, and wide
- Character operations such as character classification and code-set conversion
- String operations such as string traversal and concatenation
- Other operations on multibyte and wide characters

The following table lists the areas that you need to consider in your application to perform internationalization on character-processing tasks. These application areas appear in order of importance.

Character-Processing Task	For More Information
String traversal	“String Traversal” on page 2-30
String processing (concatenation, copying, string-length determination, character searching)	“String Processing” on page 2-32
Memory allocation	“Managing Memory for Strings and Characters” on page 2-41
Code-set conversion	“Code-Set Conversion” on page 2-22

(1 of 2)

Character-Processing Task	For More Information
Character classification	“Character Classification” on page 2-11
Case conversion	“Case Conversion” on page 2-15
Character comparison and sorting	“Character / String Comparison and Sorting” on page 2-35

(2 of 2)

Data-formatting issues such as date / time conversion are also important areas in your application to internationalize. For more information, see [Chapter 3, “Data Formatting.”](#)

Types of Characters

A GLS locale supports a particular *code set*, which maps characters to unique bit patterns called *code points*. A particular code set can contain the following types of characters:

- Single-byte characters
- Multibyte characters

Tip: For a general introduction to code sets, single-byte characters, and multibyte characters, see the “Informix Guide to GLS Functionality.”

Use of the INFORMIX-GLS library helps to remove most assumptions about the type of character that your application handles.



Single-Byte Characters

A *single-byte character* can hold code-point values from 0 to 255. It can use 7 or 8 bits of a byte to represent a character, as follows:

- The 7-bit characters make up the ASCII code set.
These characters contain code points in the range 0 to 127.
- The 8-bit characters contain code points in the range 128 to 255.
Only software that is 8-bit clean can correctly interpret 8-bit characters.

English, European, and Middle Eastern code sets support at most 256 characters. Therefore, code sets that support these languages consist of single-byte characters.

When your application processes *only* single-byte characters, it can perform string-processing tasks based on the assumption that the number of bytes in a buffer equals the number of characters that the buffer can hold. For single-byte code sets, you can rely on the built-in scaling for array allocation and access that the C compiler provides.

The INFORMIX-GLS functions and macros that handle multibyte characters are optimized for single-byte characters. Use of single-byte characters with these functions does not involve the full algorithms that multibyte-processing involves. (For more information, see [“Optimizing INFORMIX-GLS” on page 1-22.](#))

Multibyte Characters

A *multibyte character* can hold code-point values greater than 255. One multibyte character can range from 2 to 4 bytes in length. Asian code sets are multibyte code sets; they contain both single-byte and multibyte characters.

If your application processes multibyte characters, it can no longer make the same assumption as for single-byte characters. The number of bytes in a buffer no longer equals the number of characters in the buffer. Because of the potential of varying number of bytes for each character, you can no longer rely on the C compiler to perform the following operations correctly:

- Allocate space for a multibyte-character string
- Traverse a multibyte-character string
- Find the beginning of the *n*th character in a multibyte-character string

Your application cannot exploit the built-in scaling of the C compiler for multibyte-character strings, but it can use the macros and functions of the INFORMIX-GLS library to perform these operations on multibyte-character strings. To process a multibyte character, you cannot pass the entire character to a function. You must pass a pointer to the beginning of the character so that the called function can access the remaining bytes of the character.



Tip: For a list of operations that the functions of the INFORMIX-GLS library can perform on multibyte characters and multibyte-character strings, see [“Character Operations” on page 2-10](#) and [“String Operations” on page 2-30](#), respectively.

One single-byte assumption can still be applied to multibyte-character strings: no multibyte character has the null byte (0x00) as its second, third, or fourth byte. Therefore, if code is checking for only the single-byte ASCII null character, that code does not need to change to handle multibyte characters. This null character is also the null terminator in a multibyte character.



Tip: Throughout this manual, examples show how single-byte and multibyte characters appear. Because multibyte characters are usually ideographic (such as Japanese or Chinese characters), this manual does not use the actual multibyte characters. Instead, it uses ASCII characters to represent both single-byte and multibyte characters. For more information about how this manual represents multibyte and single-byte characters abstractly, see [“Character-Representation Conventions” on page 8](#) of the Introduction.

The names of most INFORMIX-GLS functions that handle multibyte characters start with one of the following strings.

Function String	What the Function Handles
ifx_gl_mb	A multibyte character
ifx_gl_mbs	A multibyte-character string

For example, **ifx_gl_mblen()** determines the length of a multibyte character, and **ifx_gl_mbslen()** determines the length of a multibyte-character string.

*The **gl_mchar_t** Data Type*

The INFORMIX-GLS library represents a multibyte character with the **gl_mchar_t** data type. The **gls.h** header file defines the **gl_mchar_t** data type, and the **ifxgls.h** header file includes **gls.h**. Therefore, you must include **ifxgls.h** in any file that uses the **gl_mchar_t** data type (or any INFORMIX-GLS function).



Important: The **gl_mchar_t** data type is an opaque structure. Do not access the individual bytes of a multibyte character directly.

Because any character in a multibyte-character string might contain up to 4 bytes, and **gl_mchar_t** refers to only *one* of those bytes, you usually declare a multibyte variable as a pointer to a **gl_mchar_t** data type. For example, the following declaration creates the **mb_string** variable as a pointer to a multibyte-character string:

```
#include <ifxgls.h>;
...
gl_mchar_t *mb_string;
```

The preceding declaration assumes that the application allocates memory for the **mb_string** multibyte string elsewhere. For information on how to allocate memory for multibyte-character strings, see [“Multibyte-Character-String Allocation” on page 2-41](#).

You can also cast a C string to a multibyte-character string, as follows:

```
char *string; /* allocated & initialized elsewhere */
...
gl_mchar_t *mbs = (gl_mchar_t *) string;
```

Wide Characters

The INFORMIX-GLS functions and macros that handle multibyte characters use special multibyte-processing algorithms to determine the size of multibyte characters. However, the overhead of these full multibyte-processing algorithms can be significant. Therefore, the INFORMIX-GLS library provides support for *wide characters* as an alternative form for the processing of multibyte characters. Wide characters allow you to rely on the C compiler built-in scaling instead of the multibyte-processing algorithms.

A wide-character form of a code set involves the normalization of the size of each multibyte character so that each character is the same size. This size must be equal to or greater than the largest character that an operating system can support, and it must match the size of an integer data type that the C compiler can scale (such as **short int**, **int**, and **long int**).

The names of most INFORMIX-GLS functions that handle wide characters start with one of the following strings.

Function String	What the Function Handles
ifx_gl_wc	A wide character
ifx_gl_wcs	A wide-character string

For example, the function **ifx_gl_wctomb()** converts a wide character to a multibyte character, and **ifx_gl_wcslen()** determines the length of a wide-character string.



The `gl_wchar_t` Data Type

The INFORMIX-GLS library represents a wide character with the **`gl_wchar_t`** data type. The **`gls.h`** header file defines the **`gl_wchar_t`** data type, and the **`ifxgls.h`** header file includes **`gls.h`**. Therefore, you must include **`ifxgls.h`** in any file that uses the **`gl_wchar_t`** data type (or any INFORMIX-GLS function).

Important: The **`gl_wchar_t`** data type is an opaque structure. Do not access the individual bytes of a wide character directly.

The **`gl_wchar_t`** data type is a fixed-length structure. Therefore, you can declare a variable as a pointer to a **`gl_wchar_t`** structure or as a **`gl_wchar_t`** structure directly. For example, the following declarations create the **`wc_string`** variable as a pointer to a wide-character string and **`wc_string2`** as a single wide character:

```
#include <ifxgls.h>;
...
gl_wchar_t *wc_string;
gl_wchar_t wc_string2;
```

The declaration of **`wc_string`** assumes that the application allocates memory for this wide-character string elsewhere. The declaration of **`wc_string2`** allocates one wide character. For information on how to allocate memory for wide-character strings, see [“Wide-Character-String Allocation” on page 2-42](#).

You can compare or assign a single-byte ASCII character or character constant to a single wide character, as in the following code fragment:

```
gl_wchar_t wc = 'a'; /* assigns an ASCII character */
if (wc=='a') /* compares with an ASCII character */
...
```

Conversion Between Multibyte and Wide Characters

To use wide characters, you convert multibyte characters to their wide-character equivalents, process the characters, and convert the wide characters back to their multibyte equivalents. The INFORMIX-GLS library supports conversion between a multibyte form of a code set and its wide-character form. Unlike code-set conversion, the actual integral value of each character does not change in this conversion.

To change all character data to wide characters, you must first locate the character data and then find all the places where it is assigned and passed to functions. INFORMIX-GLS functions perform the following tasks to convert between multibyte and wide characters:

- To convert from a multibyte character to a wide character, the code point of the multibyte character is assigned a fixed number of bytes for all values, padded on the left with zeros:
 - The **ifx_gl_mbtowc()** function performs this conversion from multibyte character to wide character.
 - The **ifx_gl_mbstowcs()** function performs this conversion from multibyte-character string to wide-character string.
- To convert from a wide character to a multibyte character, the code point of the wide character is assigned the minimum number of bytes necessary to represent the value:
 - The **ifx_gl_wctomb()** function performs this conversion from wide character to multibyte character.
 - The **ifx_gl_wcstombs()** function performs this conversion from wide-character string to multibyte-character string.

For more information on how to decide whether wide characters are appropriate for your application, see [“Processing Wide Characters” on page 1-23](#).

Character Operations

The INFORMIX-GLS library supports the following character operations on multibyte and wide characters:

- Character classification
- Case conversion
- Code-set conversion

For more information about any of the INFORMIX-GLS functions that this section describes, see the function descriptions in [Chapter 4, “INFORMIX-GLS Function Descriptions.”](#)

Character Classification

A GLS locale groups the characters of a code set into *character classes*. Each class contains characters that have a related purpose. The contents of a character class can be language specific. For example, the lower class contains all alphabetic lowercase characters in a code set. In the default locale, the default code set groups the English characters a through z into the lower class, but it also includes lowercase characters such as á, è, î, ò, and ü.

The default code set on UNIX platforms is ISO8859-1. The default code set for Windows environments is Microsoft 1252.

Tip: For more information on the default locale and the default code set, see the “Informix Guide to GLS Functionality.”

The LC_CTYPE category of a GLS locale file defines the following character classes.

Character Class	Contains
alpha	Alphabetic characters: ■ Single-byte alphabetic characters a through z and A through Z ■ Any single-byte non-English characters that the locale defines ■ Any multibyte alphabetic or digit characters that the locale defines This class includes characters in the lower and upper classes.
lower	Lowercase alphabetic characters: ■ Single-byte alphabetic characters a through z ■ Any single-byte non-English lowercase characters that the locale defines ■ Any multibyte lowercase characters that the locale defines No characters in this class are in the upper class.

(1 of 3)



Character Class	Contains
upper	Uppercase alphabetic characters: ■ Single-byte alphabetic characters A through Z ■ Any single-byte non-English uppercase characters that the locale defines ■ Any multibyte uppercase alphabetic characters that the locale defines No characters in this class are in the lower class.
digit	Single-byte decimal digits 0 through 9
xdigit	Hexadecimal digits: ■ Single-byte numeric digits 0 through 9 ■ Single-byte representations of hexadecimal digits a through f and A through F This class includes characters in the digit class.
alnum	All characters in both the alpha and digit classes.
blank	Horizontal white space: ■ Single-byte horizontal-space characters: “ ” (ASCII 0x020) tab (ASCII 0x009) ■ Any multibyte horizontal-space characters that the locale defines
space	Horizontal and vertical white space: ■ Single-byte horizontal-space characters as defined in the blank class ■ Single-byte vertical-space characters: new line, vertical tab, form feed, carriage return ■ Any multibyte vertical-space characters that the locale defines This class includes characters in the blank class.
cntrl	Control characters: ■ Single-byte control characters: ASCII 0x000 to 0x01F ■ Any other control characters that the locale defines

(2 of 3)

Character Class	Contains
graph	Graphical characters are all characters that have visual representation. This class includes characters in the alpha, lower, upper, digit, xdigit, and punct classes.
punct	Punctuation: ■ Single-byte punctuation characters: !@#\$%^&*()-+=\ '~[]{};:‘“„.?<> ■ Any non-ASCII punctuation characters that the locale defines
print	All printable characters This class includes characters in the alpha, lower, upper, digit, xdigit, graph, and punct classes.

(3 of 3)

To be internationalized, your application must not assume which characters belong in a particular character class. For example, it must not contain code such as the following example to determine whether a character is lowercase:

```
if ( one_char >= 'a' && one_char <= 'z' ) /* NOT internationalized! */
```

Instead, use functions in the INFORMIX-GLS library to identify the class of a particular character. [Figure 2-1](#) lists the GLS character classes and the INFORMIX-GLS functions that test for these classes for both multibyte and wide characters.

Figure 2-1
INFORMIX-GLS Character-Class Functions

Character Class	Multibyte-Character Function	Wide-Character Function
alnum (alpha or digit)	ifx_gl_ismalnum()	ifx_gl_iswalnum()
alpha	ifx_gl_ismalpha()	ifx_gl_iswalpha()
lower	ifx_gl_ismlower()	ifx_gl_iswlower()
upper	ifx_gl_ismupper()	ifx_gl_iswupper()
blank	ifx_gl_ismblank()	ifx_gl_iswblank()
space	ifx_gl_ismspace()	ifx_gl_iswspace()

(1 of 2)

Character Class	Multibyte-Character Function	Wide-Character Function
digit	ifx_gl_ismdigit()	ifx_gl_iswdigit()
xdigit	ifx_gl_ismxdigit()	ifx_gl_iswxdigit()
cntrl	ifx_gl_ismcntrl()	ifx_gl_iswcntrl()
graph	ifx_gl_ismgraph()	ifx_gl_iswgraph()
punct	ifx_gl_ismpunct()	ifx_gl_iswpunct()
print	ifx_gl_ismprint()	ifx_gl_iswprint()

(2 of 2)

These INFORMIX-GLS functions check the LC_CTYPE category of the current locale to determine whether a specified character belongs to the respective character classification. The following code fragment uses the **ifx_gl_ismlower()** function to perform the internationalized determination of whether a multibyte character is lowercase:

```
if ( ifx_gl_ismlower(one_char, char_size) ) /* IS internationalized! */
```

The INFORMIX-GLS functions in [Figure 2-1 on page 2-13](#) do not return a unique value if they encounter an error. To detect an error, initialize the **ifx_gl_lc_errno()** error number to 0 before you call one of these functions, and then call **ifx_gl_lc_errno()** immediately after you call the function. For example, the following code fragment performs error checking for the **ifx_gl_ismlower()** function:

```
/* Initialize the error number */
ifx_gl_lc_errno() = 0;

/* Determine if 'mb' character is lowercase */
value = ifx_gl_ismlower(mb, mb_size);

/* If the error number has changed, ifx_gl_ismlower() has
 * set it to indicate the cause of an error */
if ( ifx_gl_lc_errno() != 0 )
    /* Handle error */
else if ( value != 0 )
    /* Character 'mb' is in lower class */
else if ( value == 0 )
    /* Character 'mb' is NOT in lower class */
```

Case Conversion

In many languages, alphabetic characters have an uppercase and lowercase representation. To be internationalized, your application must not assume the case equivalent for a particular character. For example, it must not contain code such as the following to obtain the uppercase equivalent of the character in `lower_char`:

```
upper_char = lower_char - 'a' + 'A'; /* NOT internationalized! */
```

The preceding line works for the English characters of the ASCII code set. However, it does *not* work for 8-bit characters, such as à and À.

To handle case conversion in your application, use functions in the INFORMIX-GLS library to obtain the case equivalent of a particular character. [Figure 2-2](#) lists the case-conversion operations and the INFORMIX-GLS functions that perform them, both the multibyte functions and their wide-character equivalents.

Figure 2-2
INFORMIX-GLS Case-Conversion Functions

Case-Conversion Operation	Multibyte-Character Function	Wide-Character Function
Obtain the lowercase equivalent of the source character	<code>ifx_gl_tomlower()</code>	<code>ifx_gl_towlower()</code>
Obtain the uppercase equivalent of the source character	<code>ifx_gl_toupper()</code>	<code>ifx_gl_towupper()</code>

The INFORMIX-GLS case-conversion functions check the LC_CTYPE category of the current locale to determine the case equivalent of a source character. If the desired case equivalent exists, the functions return an integer value that is the alphabetic case equivalent of the source character. If no case equivalent exists, these functions return the source character.

The following code fragment uses the `ifx_gl_toupper()` function to perform the internationalized case conversion of a multibyte character:

```
ret = ifx_gl_toupper(lower_char, upper_char, char_sz) /* IS internationalized! */
```

Case Conversion for Multibyte Characters

The **ifx_gl_tolower()** and **ifx_gl_toupper()** functions require three arguments:

- The multibyte character or string to convert
- The destination buffer for the converted multibyte character or string
- The number of bytes to read to obtain a single multibyte character

For a multibyte-character string, the size of the case-converted string might not equal the size of the unconverted string. Therefore, to perform case conversion on multibyte characters, you must take the following special processing steps:

- Determine whether you need to allocate a separate destination buffer; if a destination buffer is needed, determine its size.
- Determine the number of bytes that the case-conversion process has read and written.

Determining When to Allocate a Destination Buffer

Whether you can perform case conversion of multibyte characters in place depends on whether the number of bytes written to the destination buffer is the same as the number of bytes read from the source, as follows:

- If the **ifx_gl_case_conv_outbuflen()** function determines that the size of the source string and its case-converted value are exactly equal, you can perform case conversion in place.
- If the size of the case-converted value of the source string is *not* the same as the size of the source string itself, you cannot perform case conversion in place.

If you cannot perform case conversion in place, you must allocate a separate destination buffer. To allocate this buffer, you need to have an estimate of the number of bytes that it needs to hold. Use any of the following methods to determine the number of bytes that might be written to the destination buffer:

- The **ifx_gl_case_conv_outbuflen()** function calculates either exactly the number of bytes that will be written to the destination buffer or a close overapproximation of the number.

This function applies to both uppercase and lowercase conversions. The second argument to **ifx_gl_case_conv_outbuflen()** is the number of bytes in the character source.

- The **ifx_gl_mb_loc_max()** function calculates the maximum number of bytes that can be written to the destination buffer for any source value in the *current* locale.

This value is always greater than or equal to ($\geq$) the value that the **ifx_gl_case_conv_outbuflen()** function returns.

- The macro **IFX_GL_MB_MAX** returns the maximum number of bytes that can be written to the destination buffer for any source value in *any* locale.

This value is always greater than or equal to the value that the **ifx_gl_mb_loc_max()** function returns.

Of the preceding options, the macro **IFX_GL_MB_MAX** is the fastest and the only method that can initialize static buffers. The function **ifx_gl_case_conv_outbuflen()** is the slowest but the most precise.

The following code fragment uses the **ifx_gl_mblen()** function to determine the size of the source character and the **ifx_gl_case_conv_outbuflen()** function to determine the estimated size of the case-converted value:

```
/* Obtain the sizes of the source and destination strings */
src_mb_bytes = ifx_gl_mblen(src_mb, ...);
dst_mb_bytes = ifx_gl_case_conv_outbuflen(src_mb_bytes);

if ( dst_mb_bytes == src_mb_bytes )
/* Sizes of source and case-equivalent characters are the
 * same. Perform the case conversion in place */
{
    retval =
        ifx_gl_toupper(src_mb, src_mb, IFX_GL_NO_LIMIT);
}
else
/* Sizes of source and destination characters are NOT the
 * same. Allocate a destination buffer and perform case
 * conversion into this buffer */
{
    dst_mb = (gl_mchar_t *) malloc(dst_mb_bytes);
    retval =
        ifx_gl_toupper(dst_mb, src_mb, IFX_GL_NO_LIMIT);
}
```

For more information on the **IFX_GL_NO_LIMIT** constant, see [“Multibyte-Character Termination” on page 2-39](#).

Determining the Number of Bytes Read and Written

The **ifx_gl_toupper()** and **ifx_gl_tolower()** functions return an unsigned short integer that encodes the information about the number of bytes that the function has read. The INFORMIX-GLS library provides the following macros to obtain this information from the return value.

INFORMIX-GLS Macro	Information Obtained
IFX_GL_CASE_CONV_SRC_BYTES()	The number of bytes read from the source string
IFX_GL_CASE_CONV_DST_BYTES()	The number of bytes written to the destination buffer

The following code fragment uses the **ifx_gl_tomlower()** function to convert a multibyte character to its lowercase equivalent. It uses the case-conversion macros to obtain the number of bytes read and written during the case-conversion operation:

```

/* Initialize source pointer, 'src_mb', to beginning of the
 * multibyte string. Initialize destination pointer to
 * beginning of destination buffer */
src_mb = src_mbs;
dst_mb = dst_mbs;

/* Traverse source string until the null terminator is
 * reached */
while ( *src_mb != '\0' )
{
/* Convert source multibyte character, 'src_mb', to lowercase
 * and put in the destination buffer */
    unsigned short retval =
        ifx_gl_tomlower(dst_mb, src_mb, src_mbs_bytes);
    ...
/* Increment the source pointer by the number of bytes that
 * have been read and the destination pointer by the number
 * of bytes that have been written */
    src_mb += IFX_GL_CASE_CONV_SRC_BYTES(retval);
    dst_mb += IFX_GL_CASE_CONV_DST_BYTES(retval);
    src_mbs_bytes -= IFX_GL_CASE_CONV_SRC_BYTES(retval);
}

```

The memory-management rules for case conversion of a single multibyte character also apply to converting a string of one or more multibyte characters. For example, the following code fragment converts a multibyte-character string to its uppercase equivalent:

```

/* Assume src_mbs is null terminated */
src_mbs_bytes = strlen(src_mbs);
dst_mbs_bytes = ifx_gl_case_conv_outbuflen(src_mbs_bytes);

if ( dst_mbs_bytes == src_mbs_bytes )
{
/* If two strings have the same size, overwrite each
 * multibyte character in the 'src_mbs' multibyte string
 * with its uppercase equivalent */
    src_mb = src_mbs;
}

```

```
while ( *src_mb != '\0' )
{
    retval =
        ifx_gl_toupper(src_mb,src_mb,IFX_GL_NO_LIMIT);
    src_mb += IFX_GL_CASE_CONV_SRC_BYTES(retval);
}
else
{
    /* Two strings are not the same size, so must allocate a
     * destination buffer whose size is determined by the
     * ifx_gl_case_conv_outbuflen() function */
    dst_mbs = (gl_mchar_t *) malloc(dst_mbs_bytes + 1);

    src_mb = src_mbs;
    dst_mb = dst_mbs;

    while ( *src_mb != '\0' )
    {
        retval =
            ifx_gl_toupper(dst_mb,src_mb,IFX_GL_NO_LIMIT);
        src_mb += IFX_GL_CASE_CONV_SRC_BYTES(retval);
        dst_mb += IFX_GL_CASE_CONV_DST_BYTES(retval);
    }

    *dst_mb = '\0';
}
```

Case Conversion for Wide Characters

Because a wide character has a fixed size, the **ifx_gl_towlower()** and **ifx_gl_towupper()** functions require only one argument: the wide character to convert. These functions return an integer value of the case-equivalent character for this wide character. Therefore, you can always perform case conversion of wide characters in place. For example, you can assign the case equivalent of **src_wc** back to **src_wc**, as follows:

```
src_wc = ifx_gl_towupper(src_wc);
```

You can also perform case conversion of wide characters to a destination buffer. The previous line could also be written as follows:

```
dst_wc = ifx_gl_towupper(src_wc);
```

Exception Handling

These case-conversion functions do not return a special value if they encounter an error. To detect an error, initialize the **ifx_gl_lc_errno()** error number to 0 before you call one of these functions and check **ifx_gl_lc_errno()** immediately after you call it. The following code fragment performs exception handling in the conversion of a wide character to its lowercase equivalent:

```
/* Initialize the error number */
ifx_gl_lc_errno() = 0;

/* Perform conversion of 'src_wc' to lowercase */
dst_wc = ifx_gl_towlower(src_wc);

/* If the error number has changed, ifx_gl_towlower() has set
 * it to indicate the cause of an error */
if ( ifx_gl_lc_errno() != 0 )
    /* Handle error */
else
    ...
```

Performance Issues

The INFORMIX-GLS case-conversion functions assign the destination character regardless of whether the source character has a case-equivalent character. If no case equivalent for a particular source character exists, the functions return only the source character. Therefore, the following two algorithms perform the same task:

- Calling the case-conversion function regardless of the existence of a case-equivalent character, as follows:

```
dst_wc = ifx_gl_towlower(src_wc);
```

- Calling the case-conversion function only if a case-equivalent character exists, as follows:

```
if ( ifx_gl_iswupper(src_wc) )
    dst_wc = ifx_gl_towlower(src_wc);
else
    dst_wc = src_wc;
```

However, the first approach is usually faster.

E/C

DB API

IDS/UD



Code-Set Conversion

A character might be encoded differently on two different operating systems. Therefore, the appropriate communication layer must be prepared to convert between the two encodings. This process of conversion between two code sets is called *code-set conversion*. Code-set conversion translates code points from a source code set to a destination code set.

INFORMIX-ESQL /C applications automatically perform any needed code-set conversion when they send and receive database data. ♦

The following principles apply when you send and receive database data:

- DataBlade client applications automatically perform any needed code-set conversion.
- DataBlade UDRs do not automatically perform code-set conversion. ♦

Tip: For an introduction to code-set conversion, see the “Informix Guide to GLS Functionality.”

If your application might need to perform code-set conversion, it must:

- determine if code-set conversion is needed between the source and target code sets.
- perform the code-set conversion on a character string if it is needed.

Determining If Code-Set Conversion Is Needed

The **ifx_gl_conv_needed()** function determines whether characters encoded in a source code set require conversion to a destination code set. Use this function to determine if code-set conversion is needed. Simply comparing the names of the code sets does not provide enough information to determine if it is necessary. In the **ifx_gl_conv_needed()** function, you can specify the source and destination code sets as any of the following items:

- Locale names
- Code-set names
- The IFX_GL_PROC_CS macro

For more information on these options, see [“Specifying Code-Set Names” on page 2-25](#).

Performing Code-Set Conversion

For a multibyte-character string, the size of the converted string might not equal the size of the unconverted string. Therefore, to perform code-set conversion on multibyte characters, you must take the following special processing steps:

- Determine whether you need to allocate a separate destination buffer; if a destination buffer is needed, determine its size.
- Preserve state information for multiple fragments of a multibyte string.

The following sections describe these steps.

Determining When to Allocate a Destination Buffer

Whether you can perform code-set conversion on multibyte characters in place depends on whether the number of bytes written to the destination buffer is the same as the number of bytes read from the source, as follows:

- If the **ifx_gl_cv_outbuflen()** function determines that the size of the source string and its code-set-converted value are exactly equal, you can perform code-set conversion in place.
- If the size of the code-set-converted value of the source string is *not* the same as the size of the source string itself, you cannot perform code-set conversion in place.

If you cannot perform code-set conversion in place, you must allocate a separate destination buffer. To allocate a destination buffer, you need to have an estimate of the number of bytes that it needs to hold. You can use either of the following methods to determine the number of bytes that might be written to the destination buffer:

- The **ifx_gl_cv_outbuflen()** function calculates either exactly the number of bytes that will be written to the destination buffer or a close over-approximation of the number.

The third argument to **ifx_gl_cv_outbuflen()** is the number of bytes in the character source.

- The **IFX_GL_MB_MAX** macro can be used in the following expression to calculate the maximum number of bytes that can be written to the destination buffer for any source value in *any* locale:

$(*src\_bytesleft) * IFX\_GL\_MB\_MAX$

The *src_bytesleft* value refers to the number of bytes to convert. This expression value is always greater than or equal to the expression value that uses the **ifx_gl_mb_loc_max()** function.

Of the two options, the expression that uses the macro **IFX_GL_MB_MAX** is faster and can be used to initialize static buffers. The function **ifx_gl_case_conv_outbuflen()** is slower but more precise.

The following code fragment uses the **ifx_gl_cv_outbuflen()** function to determine the estimated size of a code-set-conversion destination buffer:

```
int dstbytes;
gl_mchar_t *dstmbs;
conv_state_t state;

dstbytes = ifx_gl_cv_outbuflen("ujis", "sjis", srcbytes);
dstmbs = (gl_mchar_t *) malloc(dstbytes);

state.first_frag = 1;
state.last_frag = 1;
if (ifx_gl_cv_mconv(state, &dstmbs, &dstbytes, "ujis"
    &srcmbs, &srcbytes, "sjis") == -1 )
```

For more information on the **conv_state_t** structure, see [“Preserving State Information” on page 2-26](#).

Specifying Code-Set Names

You can specify the names of the source and destination code sets with any of the following methods:

- **Locale names**

For example, you can use **de_de.8859-1** for the German locale or **ja_jp.ujis** for the Japanese UJIS locale. For more information on locale names, see the *Informix Guide to GLS Functionality*.

- **Code-set names**

You can find the names of code sets in code-set name registry.

The code-set name registry is in **%INFORMIXDIR%\gls\cmZ** for Windows and in **\$INFORMIXDIR/gls/cmZ** for UNIX.

In the preceding pathnames, **INFORMIXDIR** is the environment variable that specifies the directory in which you install the Informix product, and **Z** represents the version number for the code-set object-file format.

- **The IFX_GL_PROC_CS macro**

This macro specifies use of the code set of the current processing locale. Depending on the context, the value of IFX_GL_PROC_CS is based on either the client environment or the database that the database server is currently accessing.

The preceding formats are valid as code-set names in any of the following INFORMIX-GLS functions:

- **ifx_gl_conv_needed()**
- **ifx_gl_cv_mconv()**
- **ifx_gl_cv_outbuflen()**
- **ifx_gl_cv_sb2sb_table()**

Preserving State Information

Most code sets are *not* state dependent; that is, the characters of these code sets can be decoded with only one algorithm, and each byte sequence represents a unique character. In contrast, byte sequences in state-dependent code sets can represent more than one character. Which character a sequence represents depends on the current state. State-dependent code sets occur primarily on IBM mainframe computers, and they affect only code-set conversion.

When you fragment a complete source string into two or more nonadjacent source buffers, you must call the **ifx_gl_cv_mconv()** function multiple times, to perform code-set conversion on each fragment of the string. Because of the nature of state-dependent code sets (and because the caller of this function cannot know whether either the source or destination code set is a state-dependent code set), you must preserve state information across the multiple calls of **ifx_gl_cv_mconv()**. The **ifx_gl_cv_mconv()** argument *state* is used for this purpose.

The *state* argument is a pointer to a **conv_state_t** structure. This structure contains two fields that you must set to indicate that you are performing code-set conversion on fragmented strings: **first_frag** and **last_frag**. The following table lists the different fragments of a string and the corresponding values to which you must set these two **conv_state_t** fields.

String Fragment	Value of first_frag Field	Value of last_frag Field
String is the first of <i>n</i> fragments.	1	0
String is the 2nd, ..., <i>n</i> th-1 fragment.	0	0
String is the last (<i>n</i> th) fragment.	0	1
String is <i>not</i> fragmented; it is a complete string.	1	1



Important: The **conv_state_t** structure contains other fields that are for internal use only. Informix does not guarantee that these other internal fields of **conv_state_t** will not change in future releases. Therefore, to create portable code, set only the **first_frag** and **last_frag** fields of the **conv_state_t** structure.

Pass the fragments to the **ifx_gl_cv_mconv()** function in the same order in which they appear in the complete string. Use the same **conv_state_t** structure for *all* of the fragments of the same complete string.

The following code performs code-set conversion on a complete character string that is *not* fragmented:

```
int unfrag_strng(out_str, out_len, out_cs, in_str,
                in_len, in_cs)
    gl_mchar_t *out_str;
    int out_len;
    char *out_cs;
    gl_mchar_t *in_str;
    int in_len;
    char *in_cs;
{
    conv_state_t state;
    int ret;

    state.first_frag = 1;
    state.last_frag = 1;
    ret = ifx_gl_cv_mconv(&state, &out_str, &out_len,
                        out_cs, &in_str, &in_len, in_cs);
    ...
}
```

This code assigns both the **first_frag** and **last_frag** fields a value of 1 to indicate that the multibyte string is *not* fragmented.

Suppose that you have a complete multibyte-character string that is fragmented into four different buffers. The following code performs code-set conversion on this fragmented string:

```
int frag_strng(out_str, out_len, out_cs, in_str,
               in_len, in_cs)
    gl_mchar_t *out_str;
    int out_len;
    char *out_cs;
    gl_mchar_t *in_str[];
    int in_len;
    char *in_cs;
{
    conv_state_t state;
    int ret;
```

```
/* Perform code-set conversion on the FIRST fragment:
 * first_frag = 1; last_frag = 0 */
state.first_frag = 1;
state.last_frag = 0;
ret = ifx_gl_cv_mconv(&state, &out_str, &out_len, out_cs,
    &in_str[0], &in_len, in_cs);
...
/* Perform code-set conversion on the SECOND fragment:
 * first_frag = 0; last_frag = 0 */
state.first_frag = 0;
state.last_frag = 0;
ret = ifx_gl_cv_mconv(&state, &out_str, &out_len, out_cs,
    &in_str[1], &in_len, in_cs);
...
/* Perform code-set conversion on the THIRD fragment.
 * No need to set the first_frag and last_frag fields again,
 * because they are already 0 */
ret = ifx_gl_cv_mconv(&state, &out_str, &out_len, out_cs,
    &in_str[2], &in_len, in_cs);
...

/* Perform code-set conversion on the FOURTH (last)
 * fragment: first_frag = 0; last_frag = 1 */
state.first_frag = 0;
state.last_frag = 1;
ret = ifx_gl_cv_mconv(&state, &out_str, &out_len, out_cs,
    &in_str[3], &in_len, in_cs);
...
}
```

For an additional issue in the processing of fragmented multibyte character strings, see [“Fragmenting Multibyte Strings” on page 2-44](#).

Performance Issues

Most performance overhead in code-set conversion is a result of either memory management or multibyte-string traversal. However, only if one of the code sets is a multibyte code set does code-set conversion require this overhead to convert correctly. If the code-set conversion is between two single-byte code sets, you can obtain a code-set *conversion table* and avoid this overhead.

The following sample code uses the **ifx_gl_cv_sb2sb_table()** function to obtain a code-set conversion table for two single-byte code sets:

```
void do_codeset_conversion(src, src_codeset, dst,
                           dst_codeset)
    unsigned char *src;
    char *src_codeset;
    unsigned char *dst;
    char *dst_codeset;
{
    unsigned char *table;

    if ( ifx_gl_cv_sb2sb_table(dst_codeset,
                               src_codeset, &table) == -1 )
        /* Handle Error */

    if ( table != NULL )
    {
        /* Convert in place */
        for ( ; *src != '\0'; src++ ) *src = table[*src];
        dst = src;
    }
    else
    {
        /* Full GLS code-set conversion, which handles
         * multibyte conversions and complex conversions
         * between single-byte code sets */
        ...
    }
}
```

String Operations

The INFORMIX-GLS library supports the following string operations on multibyte-character and wide-character strings:

- String traversal
- String processing

For more information about any of the INFORMIX-GLS functions that this section describes, see the function descriptions in [Chapter 4, “INFORMIX-GLS Function Descriptions.”](#)

String Traversal

Because a single-byte character occupies only one byte, string traversal of a single-byte character string can use the built-in scaling of the C compiler. However, you must take special steps to handle string traversal of multibyte-character and wide-character strings.

Multibyte-Character-String Traversal

Because a multibyte-character string might contain multibyte characters of different sizes, you cannot automatically traverse the string with any built-in scaling. Instead, use the following functions in INFORMIX-GLS to traverse a multibyte string:

- The **`ifx_gl_mblen()`** function determines the number of bytes in a multibyte character.
- The **`ifx_gl_mbsnext()`** function returns a pointer to the *next* multibyte character in the multibyte string.
- The **`ifx_gl_mbsprev()`** function returns a pointer to the *previous* multibyte character in the multibyte string.

These functions support string traversal in the following directions:

■ Forward

```
gl_mchar_t buf[], *p;
for ( p = buf; *p != '\0' ;
      p = ifx_gl_mbsnext(p, IFX_GL_NO_LIMIT))
    process_mchar(p);
```

■ Backward

```
gl_mchar_t buf[], *p;
p = ifx_gl_mbsprev(buf, buf + strlen(buf));
if ( p != NULL )
    for ( ; p >= buf; p = ifx_gl_mbsprev(buf, p) )
        process_mchar(p);
```

Wide-Character-String Traversal

A wide-character string consists of fixed-length characters. The INFORMIX-GLS library does not need to provide traversal functions for wide-character strings because you can traverse the string with the built-in scaling of the C compiler. However, the library does provide the **ifx_gl_wcslen()** function, which determines the number of bytes in a wide-character string.

The INFORMIX-GLS library supports wide-character string traversal in the following directions:

■ Forward

```
gl_wchar_t buf[], *p;
for ( p = buf; *p != '\0' ; p++ )
    process_wchar(*p);
```

■ Backward

```
gl_wchar_t buf[], *p;
for ( p = buf + ifx_gl_wcslen(buf) - 1; p >= buf; p-- )
    process_wchar(*p);
```

String Processing

The INFORMIX-GLS library provides functions for the following string-processing:

- Concatenation
- String copying
- String-length determination
- Character searching

The INFORMIX-GLS library provides both multibyte and wide-character functions for these string-processing tasks.

Concatenation

Concatenation is the process of appending one string to the end of another string. The following table lists the INFORMIX-GLS multibyte functions and their wide-character equivalents that perform string concatenation.

Concatenation Operation	Multibyte-Character Function	Wide-Character Function
Append a copy of one character string to the end of another character string. If the two strings overlap, the result of the operation is undefined.	ifx_gl_mbscat()	ifx_gl_wcscat()
Append a specified number of characters from one character string to the end of a second character string. If the two strings overlap, the result of this operation is undefined.	ifx_gl_mbsncat()	ifx_gl_wcsncat()

String Copying

The following table lists the INFORMIX-GLS multibyte functions and their wide-character equivalents that perform string copying.

String-Processing Task	Multibyte-Character Function	Wide-Character Function
Copy one character string to a specified location in a second character string. If the two strings overlap, the result of this operation is undefined.	ifx_gl_mbscpy()	ifx_gl_wcscpy()
Copy a specified number of characters from one character string to a second character string. If the two strings overlap, the result of this operation is undefined.	ifx_gl_mbsncpy()	ifx_gl_wcsncpy()

String-Length Determination

The following table lists the INFORMIX-GLS multibyte functions and their wide-character equivalents that determine string length or number of characters.

String-Length Task	Multibyte-Character Function	Wide-Character Function
Determine the number of characters in the string, not including any terminating null characters.	ifx_gl_mbslen()	ifx_gl_wcslen()
Determine the number of bytes in a multibyte string, not including any trailing space characters.	ifx_gl_mbsnbytes()	None
Determine the number of characters in a string, not including the trailing space characters.	ifx_gl_mbsnlen()	ifx_gl_wcsnlen()

(1 of 2)

String-Length Task	Multibyte-Character Function	Wide-Character Function
Determine the number of characters in the initial substring of one string that consists entirely of characters in a second specified character string.	ifx_gl_mbsspn()	ifx_gl_wcssp()
Determine the number of characters in the initial substring of one string that consists entirely of characters <i>not</i> in a specified second character string.	ifx_gl_mbscspn()	ifx_gl_wcscspn()

(2 of 2)

Character Searching

The following table lists the INFORMIX-GLS multibyte functions and their wide-character equivalents that perform character searching.

Character-Searching Task	Multibyte-Character Function	Wide-Character Function
Search for the first occurrence of a character in the character string.	ifx_gl_mbschr()	ifx_gl_wcschr()
Search for the last occurrence of a character in the character string.	ifx_gl_mbsrchr()	ifx_gl_wcsrchr()
Search for a substring in another string.	ifx_gl_mbsmbs()	ifx_gl_wcswcs()
Search for the first occurrence in the character string of any character from a second specified character string.	ifx_gl_mbspbrk()	ifx_gl_wcspbrk()

Character/String Comparison and Sorting

Collation involves the sorting of character data that is either stored in a database or manipulated in a client application. Informix database servers support the following methods of collation for character data:

- *Code-set collation order* refers to the bit-pattern order of characters within a code set.
The order of the code points in the code set determines the sort order.
- *Localized collation order* refers to an order of the characters that relates to a real language.

The LC_COLLATE category of a GLS locale file defines the order of the characters in the localized order.



Tip: For more information on code-set and localized collation order, see the “Informix Guide to GLS Functionality.”

To perform code-set collation, you compare only the integer values of two multibyte or two wide characters. For example, suppose one multibyte character, **mbs1**, contains the value $A^1A^2A^3$ and a second multibyte character, **mbs2**, contains the value $B^1B^2B^3$. If the integer value of $A^1A^2A^3$ is less than the integer value of $B^1B^2B^3$, then **mbs1** is less than **mbs2** in code-set collation order.

However, sometimes you want to sort character data according to the language usage of the characters. In code-set order, the character *a* is greater than the character *A*. In many contexts, you would probably not want the string *Apple* to sort *before* the string *apple*. The localized order could list the character *A* after the character *a*. Similarly, even though the character *À* might have a code point of 133, the localized order could list this character after *A* and before *B* (*A*=65, *À*=133, *B*=66). In this case, the string *ÀB* sorts after *AC* but before *BD*.

The following table lists the INFORMIX-GLS functions that use localized collation order to compare two multibyte-character or wide-character strings.

String-Comparison Task	Multibyte-Character Function	Wide-Character Function
Compare two character strings by localized collation order.	ifx_gl_mbscoll()	ifx_gl_wcscoll()

These functions access the LC_COLLATE category of a locale file to obtain localized collating information when they compare character strings. They return an integer value that indicates the results of the comparison between two string arguments. The following table shows the comparison between the first string argument (Arg 1) and the second string argument (Arg 2), as well as the return values.

Argument Comparison	Return Value
Arg 1 < Arg 2	<0
Arg 1 = Arg 2	0
Arg 1 > Arg 2	>0

The **ifx_gl_mbcoll()** and **ifx_gl_wcscoll()** functions do *not* return a special value if an error has occurred. Therefore, to detect an error condition, you must initialize the **ifx_gl_lc_errno()** error number to zero *before* you call one of these functions and check **ifx_gl_lc_errno()** after you call the function. The following code fragment performs error checking for a call to the **ifx_gl_wcscoll()** function:

```
/* Initialize the error number */
ifx_gl_lc_errno() = 0;

/* Compare the two wide-character strings */
value = ifx_gl_wcscoll(wcs1, wcs1_char_length, wcs2,
                      wcs2_char_length);

/* If the error number has changed, ifx_gl_wcscoll() has
 * set it to indicate the cause of an error */
if ( ifx_gl_lc_errno() != 0 )
    /* Handle error */
else if ( value < 0 )
    /* wcs1 is less than wcs2 */
else if ( value == 0 )
    /* wcs1 is equal to wcs2 */
else if ( value > 0 )
    /* wcs1 is greater than wcs2 */
...
```

Other Operations

In addition to the operations on characters and strings, the INFORMIX-GLS library provides support for the following operations:

- Terminating characters and strings
- Allocating memory for characters and strings
- Truncating and fragmenting strings

String and Character Termination

You can use the INFORMIX-GLS library with many different application programming interfaces (APIs), which might handle strings in different ways. To provide flexible support for APIs, the INFORMIX-GLS library allows you to indicate how to handle the following items:

- Character-string termination
Is the string argument a null-terminated string?
- Multibyte-character termination
Is the length of a multibyte character known?

Character-String Termination

The API that you use with the INFORMIX-GLS library might handle string termination in either of the following ways:

- All character strings are terminated with a null character.
The null character indicates the end of the string. Such strings are called null-terminated strings. The null terminator of a multibyte string consists of one byte whose value is 0. The null terminator of a wide-character string consists of one **gl_wchar_t** character whose value is 0.
- Each string consists of a pointer and length that indicates the number of bytes in the string.
Character strings that are *not* null-terminated are called *length-terminated strings*. Length-terminated strings can contain null characters, but these null characters do not indicate the end of the string.

The INFORMIX-GLS functions that take a string argument allow you to pass this string as either a null-terminated string or a length-terminated string. To provide this flexibility, many INFORMIX-GLS functions that accept a multibyte or wide-character string expect the string to be represented with the following two arguments:

- The string itself
- The length of the string

The value that you provide for the string length tells the INFORMIX-GLS function how to handle the associated string, as the following table shows.

String-Length Value	Meaning
IFX_GL_NULL	The INFORMIX-GLS function assumes that the string is a null-terminated string.
>=0	The INFORMIX-GLS function assumes that this length indicates the number of bytes in the length-terminated string.
<0, != IFX_GL_NULL	The INFORMIX-GLS function sets the error number to the IFX_GL_PARAMERR error.

Multibyte-Character Termination

Many GLS library functions operate on just one multibyte character. Each INFORMIX-GLS function that accepts a multibyte character expects the character to be represented by the following two arguments:

- The character itself
- The maximum length of the character

The value that you provide for the character length tells the INFORMIX-GLS function how to handle the associated character, as the following table shows.

String-Length Value	Meaning
IFX_GL_NO_LIMIT	The INFORMIX-GLS function reads as many bytes as necessary from the multibyte character to form a complete character.
>=0	The INFORMIX-GLS function does <i>not</i> read more than this number of bytes from the multibyte character when it tries to form a complete character.
<0, != IFX_GL_NO_LIMIT	The INFORMIX-GLS function sets the error number to the IFX_GL_EINVAL error.

If the multibyte character is in a null-terminated multibyte string, the character length must be `IFX_GL_NO_LIMIT`. For example, if **mbs** points to a null-terminated string of multibyte characters, the following code fragment must specify `IFX_GL_NO_LIMIT` as the character length:

```
for ( mb = mbs; *mb != '\0'; mb += bytes )
{
    if ( (bytes = ifx_gl_mblen(mb, IFX_GL_NO_LIMIT)) == -1 )
        /* handle error */
}
```

If a multibyte character, **mb**, is in a length-terminated multibyte string or is a character in a buffer by itself, the character length must equal the number of bytes between where **mb** points and the end of the buffer that holds the string or character. For example, if **mbs** points to a length-terminated string of multibyte characters and **mbs_bytes** is the number of bytes in that string, the following call to **ifx_gl_mblen()** must specify the length of the multibyte string:

```
for ( mb = mbs; mbs_bytes > 0; mb += bytes,
      mbs_bytes -= bytes )
{
    if ( (bytes = ifx_gl_mblen(mb, mbs_bytes)) == -1 )
        /* handle error */
}
```

Similarly, if **mb** points to one multibyte character and **mb_bytes** is the number of bytes in the buffer that holds the character, the following call to **ifx_gl_mblen()** must specify the length of the multibyte character:

```
if ( (bytes = ifx_gl_mblen(mb, mb_bytes)) == -1 )
    /* handle error */
```

If the INFORMIX-GLS function cannot determine whether bytes in a buffer make up a valid multibyte character, it sets the error number to `IFX_GL_EINVAL`. The function is unable to determine a valid multibyte character for the following reasons:

- The function would need to read more than the specified number of bytes to recognize a multibyte character.
- The specified character length is less than or equal to zero (≤ 0).

Tip: Wide characters are fixed length. Therefore, INFORMIX-GLS functions that operate on wide characters do not require the character length.



Managing Memory for Strings and Characters

You must make buffers large enough to hold text in any of the languages that your application will handle. If your application will handle many languages, you must ensure that allocated buffers are large enough to hold translated versions of the text. If your application will handle Asian (multibyte) languages, you need to replace single-byte buffers with multibyte- or wide-character buffers.



Important: Any memory that INFORMIX-GLS functions allocate remains allocated only for the duration of the function. It does not remain after the function returns. Therefore, you must manage memory for multibyte-character and wide-character strings.

Multibyte-Character-String Allocation

Multibyte characters have varying lengths. When you represent a multibyte-character string in an array, the number of array elements does *not* equal the number of multibyte characters in the string. Therefore, you cannot use the same allocation method for multibyte strings as for single-byte strings.

Instead, you can use the following INFORMIX-GLS macro and functions to help you determine how much memory a multibyte character requires.

INFORMIX-GLS Function	Purpose
IFX_GL_MB_MAX	Indicates the maximum number of bytes that any multibyte character in <i>any</i> locale can occupy. Use this macro to allocate space in static buffers that are intended to contain one or more multibyte characters.
ifx_gl_mb_loc_max()	Returns the maximum number of bytes that any character in the current locale can occupy.
ifx_gl_cv_outbuflen(), ifx_gl_case_conv_outbuflen()	Calculates one of the following values: ■ Exactly the number of bytes that are required by a destination buffer of the converted multibyte characters ■ A close overapproximation of the number

For example, the following declaration statically allocates 20 multibyte characters for the **mbs** string:

```
gl_mchar_t mbs[20 * IFX_GL_MB_MAX]; /* static allocation */
```

The following declarations dynamically allocate 20 multibyte characters for the **mb1** and **mb2** strings:

```
gl_mchar_t *mbs1 = (gl_mchar_t *) malloc(20*IFX_GL_MB_MAX);  
gl_mchar_t *mbs2 = (gl_mchar_t *) malloc(20*ifx_gl_mb_loc_max());
```

The declaration for **mb1** uses the maximum multibyte-character size. The declaration for **mb2** uses the **ifx_gl_mb_loc_max()** function to obtain a more precise estimate for the size of 20 multibyte characters. The **ifx_gl_mb_loc_max()** function returns the maximum size among all characters in the *current* processing locale.

If your multibyte-character string is null terminated, allocate one additional byte for the null terminator. The following declarations allocate three null-terminated multibyte-character strings:

```
/* static allocation */  
gl_mchar_t mbs[20*IFX_GL_MB_MAX+1];  
  
/* dynamic allocation with IFX_GL_MB_MAX */  
gl_mchar_t *mbs1 = (gl_mchar_t *) malloc(20*IFX_GL_MB_MAX+1);  
  
/* dynamic allocation with ifx_gl_mb_loc_max() */  
gl_mchar_t *mbs2 = (gl_mchar_t *)  
    malloc(20*ifx_gl_mb_loc_max()+1);
```

Wide-Character-String Allocation

When you represent a wide-character string in an array, the number of array elements *does* equal the number of wide characters in the string. Therefore, you can use the same allocation method for wide-character strings as for single-byte strings. For example, the following declaration statically allocates 20 wide characters for the **wcs** string:

```
gl_wchar_t wcs[20];
```

The following declaration dynamically allocates 20 wide characters for the **wc1** string:

```
gl_wchar_t *wcs1 = (gl_wchar_t *) malloc(20*sizeof(gl_wchar_t));
```

If your wide-character string is null terminated, you must allocate one additional character for the null terminator. The null terminator requires the same space allocated as an entire wide-character. The following declaration allocates three null-terminated wide-character strings:

```
/* static allocation */
gl_wchar_t wcs[21];
/* dynamic allocation */
gl_wchar_t *wcs3 = (gl_wchar_t *) malloc(21*sizeof(gl_wchar_t));
```

String Deallocation

The INFORMIX-GLS library does *not* automatically deallocate memory that you dynamically allocate. Once you no longer need the string buffer, you must ensure that you deallocate any memory that your application has dynamically allocated for multibyte-character and wide-character strings.

DB API

The DataBlade API does provide some automatic garbage collection for memory that you allocate dynamically. When this memory is deallocated depends on the memory duration with which it was allocated. However, it is good programming practice to handle memory deallocation implicitly whenever possible. For more information on memory management with the DataBlade API, see the *DataBlade API Programmer's Manual*. ♦

Keeping Multibyte Strings Consistent

You must take special measures to perform the following operations on multibyte strings so that you do not split a multibyte character:

- *Truncation* makes a long character string fit into a smaller buffer.
- *Fragmentation* breaks a long character string into two or more non-adjacent buffers to meet the memory-management requirements of their components.

Truncating Multibyte Strings

Sometimes you need to truncate a long character string so that it fits into a smaller buffer. When you truncate a character string that contains just single-byte characters, you can truncate at an arbitrary byte location in the string. Because each character is one byte long, the truncated result still contains only complete characters.

However, to truncate a string that might contain even one multibyte character, you must take special measures. If you truncate at an arbitrary byte location in a multibyte-character string, you might truncate at a byte that is part of a multibyte character. In this case, the truncated string might end with only the first 1, 2, or 3 bytes of a multibyte character without the remaining bytes of the character. For such a string, subsequent traversal could result in an attempt to read beyond the end of the buffer.

Therefore, all INFORMIX-GLS functions that traverse one multibyte character or a length-terminated multibyte-character string set the error number to `IFX_GL_EINVAL` if they detect that an otherwise valid character has been truncated.

If you know that no truncation has occurred to the string, you can consider the `IFX_GL_EINVAL` error the same as `IFX_GL_EILSEQ`. However, if truncation might have occurred, `IFX_GL_EINVAL` indicates that you need to further truncate the string so that the last character in the string is complete. Depending on your application, you might take one of the following actions:

- Make the truncated string even shorter than originally intended.
- Replace the first 1, 2, or 3 bytes of the truncated character with a padding character that is appropriate for your application.



Important: *Even though the INFORMIX-GLS library functions can detect invalid characters after truncation has occurred, it is much better to avoid the situation.*

Fragmenting Multibyte Strings

Sometimes you need to fragment a long character string into two or more nonadjacent buffers to meet the memory-management requirements of their components. When you fragment a character string that contains just single-byte characters, you can fragment at an arbitrary byte location in the string. Because each character is one byte long, the fragmented results are still each a complete character string.

However, to fragment a string that might contain even one multibyte character, you must take special measures. If you fragment at an arbitrary byte location in a multibyte-character string, you might fragment at a byte that is part of a multibyte character. In this case, one fragment might end with the first 1, 2, or 3 bytes of a character, while the next fragment starts with the remaining byte or bytes.

If the only thing that you ever do with these fragments is to concatenate them back together to form one string, you do not need to perform any special processing. However, if you need to traverse the fragments as multibyte strings, these fragments might cause an attempt to read beyond the end of one fragment or an illegal character at the beginning of the next fragment.

Therefore, all INFORMIX-GLS functions that traverse one multibyte character or a length-terminated multibyte-character string set the error number to IFX_GL_EINVAL if they detect an otherwise valid character at the end of a fragment.



Important: *The INFORMIX-GLS functions cannot detect that the beginning of a fragment contains the remaining bytes of the last character in some previous fragment because they cannot look at the previous fragment first. Therefore, they might interpret the last 1, 2, or 3 bytes of a multibyte character as a valid character.*

If you know that no fragmentation has occurred on the string, you can consider the IFX_GL_EINVAL error the same as IFX_GL_EILSEQ. However, if fragmentation might have occurred, IFX_GL_EINVAL indicates that you need to fragment the string so that each fragment is a complete string. Depending upon your application, you might take one of the following actions:

- Make a fragment even shorter than originally intended.
- Replace the first 1, 2, or 3 bytes of the fragmented character with a padding character that is appropriate for your application and shift these bytes to the beginning of the next fragment.



Important: *Even though the INFORMIX-GLS library functions can detect invalid characters after fragmentation has occurred, it is much better to avoid the situation.*

Data Formatting

Locale-Specific Data Formats	3-3
LC_NUMERIC Locale-File Category	3-4
LC_MONETARY Locale-File Category.	3-5
LC_TIME Locale-File Category	3-6
Conversion and Formatting with INFORMIX-GLS	3-8
Converting a Locale-Specific String	3-10
Formatting a Locale-Specific String	3-11

In This Chapter

When you internationalize an application, you need to consider how to handle the format of locale-specific data. This chapter answers the following questions:

- What are the different locale-file categories and what locale-specific formats do they define?
- How do you use the INFORMIX-GLS library to convert and format locale-specific data?

Character-processing issues, such as how to handle multibyte strings, are important areas in your application to internationalize. For more information, see [Chapter 2, “Character Processing.”](#)

Locale-Specific Data Formats

The format in which data appears within an application when it is in literal strings or character variables is called the *end-user format*. End-user formats are locale specific: different countries use different formats for numeric, monetary, date, and date / time data. The GLS locale file defines locale-specific formats for each of these types of data, as the following table shows.

Type of Data	Locale-File Category	SQL Built-In Data Types
Numeric	LC_NUMERIC	DECIMAL, INT8, INTEGER, SMALLINT, FLOAT, SMALLFLOAT
Monetary	LC_MONETARY	MONEY
Date and Time	LC_TIME	DATE, DATETIME



Tip: For more information on the end-user formats that this chapter describes, see the “Informix Guide to GLS Functionality.”

LC_NUMERIC Locale-File Category

The LC_NUMERIC locale-file category is the section within a GLS locale file that defines the locale-specific formats for strings that contain numeric data. Numeric data is considered to be all values that contain digits *except* monetary data. Therefore, integer, fixed-point, and floating-point numbers are all considered numeric data. Strings of numeric data are called *number strings*.



Tip: In the “Informix Guide to GLS Functionality,” the LC_NUMERIC locale-file category is sometimes referred to as the NUMERIC locale category. This manual uses LC_NUMERIC as the category name for numeric data.

The LC_NUMERIC category of each locale file contains subcategories that define the different numeric formats for that locale. The following table lists the numeric formats and their corresponding LC_NUMERIC subcategory.

Numeric Format	LC_NUMERIC Subcategory
Numeric decimal separator	decimal_point
Numeric thousands separator	thousands_sep
Numeric grouping information (number of numeric digits to group together before inserting a thousands separator)	grouping
Numeric positive sign	num_positive_sign
Numeric negative sign	num_negative_sign

LC_MONETARY Locale-File Category

The LC_MONETARY locale-file category is the section within a GLS locale file that defines the locale-specific formats for strings that contain monetary data. Monetary data is considered to be values that contain digits and that represent units of some currency. Therefore, only fixed-point numbers can be monetary data. All other types of numbers (such as integer and floating-point) are considered to be numeric data. Strings of monetary data are called *money strings*.



Tip: In the “Informix Guide to GLS Functionality,” the LC_MONETARY locale-file category is sometimes referred to as the MONETARY locale category. This manual uses LC_MONETARY as the category name for monetary data.

The LC_MONETARY category of each locale file contains subcategories that define the different monetary formats for that locale. The following table lists the monetary formats and their corresponding LC_MONETARY subcategory.

Monetary Format	LC_MONETARY Subcategory
International currency symbol (in accordance with the ISO 4217:1987 standard)	int_curr_symbol
National (local) currency symbol	currency_symbol
Monetary decimal separator	mon_decimal_point
Monetary thousands separator	mon_thousands_sep
Monetary grouping information (number of monetary digits to group together before inserting a thousands separator)	mon_grouping
Monetary positive sign	positive_sign
Monetary negative sign	negative_sign
Number of fractional digits (those to the right of the decimal separator) to use in the international monetary format	int_frac_digits
Number of fractional digits (those to the right of the decimal separator) to use in the national (local) monetary format	frac_digits

(1 of 2)



Monetary Format	LC_MONETARY Subcategory
Whether the currency symbol precedes or follows the monetary value	p_cs_precedes, n_cs_precedes
Whether to separate the currency symbol from the monetary value with a space	p_sep_by_space, n_sep_by_space
Position of positive and negative signs in a monetary value	p_sign_posn, n_sign_posn

(2 of 2)

Tip: The **DBMONEY** environment variable can also specify formats for locale-specific formats for money strings.

For definitions of any of these monetary formats (such as currency symbol, decimal separator, or thousands separator) and more information on the **DBMONEY** environment variable, see the *Informix Guide to GLS Functionality*.

LC_TIME Locale-File Category

The LC_TIME locale-file category is the section within a GLS locale file that defines the locale-specific formats for strings that contain date and time data. Such strings are called *date strings* (which contain only date data) or *date/time strings* (which contain date and time data).



Tip: In the “*Informix Guide to GLS Functionality*,” the LC_TIME locale-file category is sometimes referred to as the *TIME* locale category. This manual uses LC_TIME as the category name for date and time data.

The LC_TIME category of each locale file contains subcategories that define the different date and time formats. The following table lists the available formats and their corresponding LC_TIME subcategory.

Date/Time Format	LC_TIME Subcategory
Abbreviated weekday names	abday
Full weekday names	day
Abbreviated month names	abmon
Full month names	mon
Date representation	d_fmt
Time representation	t_fmt
Date / time representation	d_t_fmt
A.M. and P.M. equivalents	am_pm
Representation of date / time with A.M. and P.M. indicators	t_fmt_ampm
Era names	era
Representation of date with eras	era_d_fmt
Representation of time with eras	era_t_fmt
Representation of date / time with eras	era_d_t_fmt
Alternative digits to use in date and time strings	alt_digits



Tip: The **GL_DATE**, **GL_DATETIME**, **DBDATE**, and **DBTIME** environment variables can also specify formats for locale-specific formats for date and time strings.

For definitions of any of these date or time formats (such as eras) and information on related environment variables, see the *Informix Guide to GLS Functionality*.

Conversion and Formatting with INFORMIX-GLS

The INFORMIX-GLS library provides functions that allow you to perform the following tasks on locale-specific data:

- *Conversion* changes a string that contains locale-specific formats to the internal representation of its value.
You usually perform conversion on a locale-specific string to prepare it for storage in a program variable or a database column.
- *Formatting* changes the internal representation of a value to a locale-specific string.
You usually perform formatting of a locale-specific string to prepare the internal representation of a value for display or printing to the end user.

The internal representation of a value is one that can be stored directly in a database column. The following table lists the SQL data types for the internal representations along with the ESQL/C and DataBlade API data types that hold these internal representations.

Type of Data	SQL Data Type	ESQL/C Data Type	DataBlade API Data Type
Numeric	DECIMAL * SMALLINT, INTEGER, INT8, FLOAT, SMALLFLOAT	decimal (dec_t)	mi_decimal
Monetary	MONEY	decimal (dec_t)	mi_decimal
Date	DATE	date (long int)	mi_date
Date and time	DATETIME	datetime (dtime_t)	mi_datetime

* Numeric data includes integer, floating-point, and fixed-point numbers. The INFORMIX-GLS conversion and formatting functions represent all these numeric data types with the **decimal** (or **mi_decimal**) data type. To convert the **decimal** (or **mi_decimal**) representation to an integer or floating-point value, use a library function such as **dectoint()**, **dectolong()**, or **dectodbl()**. For more information on these decimal-conversion functions, see the *INFORMIX-ESQL/C Programmer's Manual*.

Figure 3-1 lists the functions that the INFORMIX-GLS library provides for the conversion and formatting of locale-specific data.

Figure 3-1
INFORMIX-GLS Conversion and Formatting Functions

Locale-Specific Data	Conversion	Formatting
Numeric	<code>ifx_gl_convert_number()</code>	<code>ifx_gl_format_number()</code>
Monetary	<code>ifx_gl_convert_money()</code>	<code>ifx_gl_format_money()</code>
Date	<code>ifx_gl_convert_date()</code>	<code>ifx_gl_format_date()</code>
Date / time	<code>ifx_gl_convert_datetime()</code>	<code>ifx_gl_format_datetime()</code>

The conversion and formatting functions in Figure 3-1 accept a *format string* to indicate how to handle a locale-specific string. A format string is composed of white spaces, ordinary characters, and one or more formatting directives. A *formatting directive* consists of the following characters:

- A percent symbol (%)
- Optional format modifiers
- A type-specifier character, which determines the type of conversion

For example, the `ifx_gl_convert_number()` function supports the following formatting directive:

`%0x`

In the preceding formatting directive, 0 is a format modifier that indicates padding, and x is a type specifier that indicates the hexadecimal format of a number.

Converting a Locale-Specific String

The INFORMIX-GLS conversion functions scan the incoming locale-specific string to create a corresponding internal representation, as follows.

Conversion Function	Unconverted Form	Converted Form	
		ESQL/C Data Type	DataBlade API Data Type
<code>ifx_gl_convert_number()</code>	Number string	decimal (dec_t)	mi_decimal
<code>ifx_gl_convert_money()</code>	Money string	decimal (dec_t)	mi_decimal
<code>ifx_gl_convert_date()</code>	Date string	date (long int)	mi_date
<code>ifx_gl_convert_datetime()</code>	Date / time string	datetime (dttime_t)	mi_datetime

The formatting directives in the format string tell the conversion functions what conversions to perform on the locale-specific string. To process a formatting directive, a conversion function consults the appropriate category in the current locale to obtain any locale-specific formats and then converts the resulting value to an internal representation that can be stored in a database.

For example, if the current locale is the default locale, the following DataBlade API call to the **ifx_gl_convert_number()** function converts the number string "1,450" to its hexadecimal equivalent in an **mi_decimal** value:

```
mi_decimal num_val;
...
if ( ifx_gl_convert_number(&num_val, "1,450", "%0x") != 0 )
    /* handle error */
```

In the current locale, the thousands separator is defined as the comma (,) symbol. Therefore, the **ifx_gl_convert_number()** function must correctly interpret the comma in the number string so that it can convert this string to the hexadecimal equivalent of the value 1450 and store the result in the **mi_decimal** value, **num_val**.

If the current locale is French (**fr_fr**), the thousands separator is a space. Therefore, the following call to **ifx_gl_convert_number()** must interpret a space as the thousands separator to convert the French number string ("1 450") successfully to its **mi_decimal** equivalent:

```
mi_decimal *num_val;
...
if ( ifx_gl_convert_number(num_val, "1 450", "%0x") != 0 )
    /* handle error */
```

Formatting a Locale-Specific String

The INFORMIX-GLS formatting functions create a locale-specific string from an incoming internal representation of a value, as follows.

Formatting Function	Unformatted Form		Formatted Form
	ESQL/C Internal Representation	DataBlade API Internal Representation	
ifx_gl_format_number()	decimal (dec_t)	mi_decimal	Number string
ifx_gl_format_money()	decimal (dec_t)	mi_decimal	Money string
ifx_gl_format_date()	date (long int)	mi_date	Date string
ifx_gl_format_datetime()	datetime (dtime_t)	mi_datetime	Date / time string

The formatting directives in the format string tell the formatting functions how to format the internal representation of a value into a locale-specific string. To process a formatting directive, a formatting function consults the appropriate category in the current locale to obtain any locale-specific formats and then formats the locale-specific string with this information.

For example, the following DataBlade API call to the function **ifx_gl_format_number()** converts the hexadecimal representation of the number 1450 to a number string:

```
mi_decimal num_val;  
char num_str[BUFSIZE];  
...  
if ( ifx_gl_format_number(&num_str, BUFSIZE, num_val, "%0x") != 0 )  
    /* handle error */
```

If the current locale is the default locale, **num_str** contains the number string "1,450" upon successful completion of this call to **ifx_gl_format_number()**. The function correctly formats the comma (,) as the thousands separator in the number string. If the current locale is French (**fr_fr**), the thousands separator is a space. Therefore, the preceding call to the function **ifx_gl_convert_number()** would format the French number string as "1 450".

INFORMIX-GLS Function Descriptions

Function Summary	4-5
Memory Allocation	4-5
Initialization and Error Handling	4-6
Stream Input and Output	4-6
Character Processing	4-6
Code-Set Conversion	4-7
Character Classification	4-7
Case Conversion	4-8
String Traversal	4-8
String Processing	4-8
Character/String Comparison and Sorting	4-9
Data Formatting	4-9
Date and Time Conversion and Formatting Functions	4-10
Numeric Conversion and Formatting	4-10
Money Conversion and Formatting	4-10
Function Reference	4-10
ifx_gl_case_conv_outbuflen()	4-11
ifx_gl_conv_needed().	4-13
ifx_gl_convert_date().	4-15
ifx_gl_convert_datetime()	4-24
ifx_gl_convert_money().	4-35
ifx_gl_convert_number()	4-40
ifx_gl_cv_mconv().	4-45
ifx_gl_cv_outbuflen().	4-49
ifx_gl_cv_sb2sb_table()	4-52
ifx_gl_format_date()	4-54
ifx_gl_format_datetime()	4-63
ifx_gl_format_money()	4-74

ifx_gl_format_number()	4-80
ifx_gl_getmb()	4-86
ifx_gl_init()	4-89
ifx_gl_ismalnum()	4-91
ifx_gl_ismalpha()	4-93
ifx_gl_ismblank()	4-96
ifx_gl_ismcntrl()	4-98
ifx_gl_ismdigit()	4-100
ifx_gl_ismgraph()	4-102
ifx_gl_ismlower()	4-104
ifx_gl_ismprint()	4-106
ifx_gl_ismpunct()	4-108
ifx_gl_ismspace()	4-110
ifx_gl_ismupper()	4-113
ifx_gl_ismxdigit()	4-115
ifx_gl_iswalnum()	4-118
ifx_gl_iswalpha()	4-120
ifx_gl_iswblank()	4-122
ifx_gl_iswcntrl()	4-124
ifx_gl_iswdigit()	4-126
ifx_gl_iswgraph()	4-128
ifx_gl_iswlower()	4-130
ifx_gl_iswprint()	4-132
ifx_gl_iswpunct()	4-134
ifx_gl_iswspace()	4-136
ifx_gl_iswupper()	4-138
ifx_gl_iswxdigit()	4-140
ifx_gl_lc_errno()	4-142
ifx_gl_mb_loc_max()	4-144
ifx_gl_mblen()	4-145
ifx_gl_mbscat()	4-147
ifx_gl_mbschr()	4-150
ifx_gl_mbscoll()	4-152
ifx_gl_mbscpy()	4-155
ifx_gl_mbscspn()	4-157
ifx_gl_mbslen()	4-160

ifx_gl_mbsmbs()		.4-162
ifx_gl_mbsncat()		.4-165
ifx_gl_mbsncpy()		.4-168
ifx_gl_mbsnext()		.4-171
ifx_gl_mbsntsbytes()		.4-173
ifx_gl_mbsntslen()		.4-175
ifx_gl_mbspbrk()		.4-177
ifx_gl_mbsprev()		.4-180
ifx_gl_mbsrchr()		.4-182
ifx_gl_mbsspn()		.4-185
ifx_gl_mbstowcs()		.4-188
ifx_gl_mbtowc()		.4-191
ifx_gl_putmb()		.4-193
ifx_gl_tolower()		.4-196
ifx_gl_toupper()		.4-199
ifx_gl_towlower()		.4-202
ifx_gl_towupper()		.4-204
ifx_gl_wcscat()		.4-206
ifx_gl_wcschr()		.4-208
ifx_gl_wcscoll()		.4-210
ifx_gl_wcscpy()		.4-213
ifx_gl_wscspn()		.4-215
ifx_gl_wcslen()		.4-218
ifx_gl_wcsncat()		.4-220
ifx_gl_wcsncpy()		.4-222
ifx_gl_wcsntslen()		.4-224
ifx_gl_wcspbrk()		.4-226
ifx_gl_wcsrchr()		.4-228
ifx_gl_wcssp()		.4-230
ifx_gl_wcstombs()		.4-233
ifx_gl_wcswcs()		.4-235
ifx_gl_wctomb()		.4-237



DB API

In This Chapter

This chapter describes the functions that the INFORMIX-GLS library provides. First is a summary of functions by categories. The rest of this chapter provides reference information about the INFORMIX-GLS public functions, in alphabetical order.

Tip: The INFORMIX-GLS manual pages are also available on-line as HTML pages. The HTML pages reside in a subdirectory of **gls** under the **INFORMIXDIR** directory.

For information about DataBlade API header files and data structures to use with the functions, see the *DataBlade API Programmer's Manual*. ♦

Function Summary

This section groups all the functions of the INFORMIX-GLS library by task. For information about an individual function, see its corresponding function description in “[Function Reference](#),” which begins on [page 4-10](#). For a general introduction to a task, refer to the other chapters of this manual.

Memory Allocation

The INFORMIX-GLS library provides the following macro and function to assist in memory allocation:

- The `IFX_GL_MB_MAX` macro
- The `ifx_gl_mb_loc_max()` function

For more information, see “[Allocating Memory](#)” on [page 1-21](#).

Initialization and Error Handling

The INFORMIX-GLS library provides the following functions:

- The **ifx_gl_init()** function initializes the INFORMIX-GLS library.
For more information, see [“Initializing the INFORMIX-GLS Library” on page 1-18.](#)
- The **ifx_gl_lc_errno()** function obtains the value of the INFORMIX-GLS error number.
For more information, see [“Handling INFORMIX-GLS Exceptions” on page 1-19.](#)

Stream Input and Output

The INFORMIX-GLS library provides the following functions to input and output multibyte characters in a user-defined location:

- **ifx_gl_getmb()**
- **ifx_gl_putmb()**

For more information, see [“Using Stream Input and Output” on page 1-21.](#)

Character Processing

Most of the INFORMIX-GLS functions provide support for character processing of both multibyte and wide characters. This section lists the character-processing tasks and the INFORMIX-GLS functions that support them. For general information on character processing, see [Chapter 2, “Character Processing.”](#)

Code-Set Conversion

The INFORMIX-GLS library provides the following functions to perform code-set conversion:

- **ifx_gl_conv_needed()**
- **ifx_gl_cv_mconv()**
- **ifx_gl_cv_outbuflen()**
- **ifx_gl_cv_sb2sb_table()**

For general information, see [“Code-Set Conversion” on page 2-22](#).

Character Classification

The INFORMIX-GLS library provides the following functions for character classification of multibyte and wide characters.

Multibyte-Character Classification	Wide-Character Classification
ifx_gl_ismalnum()	ifx_gl_iswalnum()
ifx_gl_ismalpha()	ifx_gl_iswalpha()
ifx_gl_ismblank()	ifx_gl_iswblank()
ifx_gl_ismcntrl()	ifx_gl_iswcntrl()
ifx_gl_ismdigit()	ifx_gl_iswdigit()
ifx_gl_ismgraph()	ifx_gl_iswgraph()
ifx_gl_ismlower()	ifx_gl_iswlower()
ifx_gl_ismprint()	ifx_gl_iswprint()
ifx_gl_ismpunct()	ifx_gl_iswpunct()
ifx_gl_ismspace()	ifx_gl_iswspace()
ifx_gl_ismupper()	ifx_gl_iswupper()
ifx_gl_ismxdigit()	ifx_gl_iswxdigit()

For general information, see [“Character Classification” on page 2-11](#).

Case Conversion

The INFORMIX-GLS library provides the following functions for case conversion of multibyte and wide characters.

Multibyte-Character Case Conversion	Wide-Character Case Conversion
<code>ifx_gl_tomlower()</code>	<code>ifx_gl_towlower()</code>
<code>ifx_gl_toupper()</code>	<code>ifx_gl_toupper()</code>
<code>ifx_gl_mbsspn()</code>	<code>ifx_gl_wcssp()</code>

In addition, the INFORMIX-GLS library provides the function `ifx_gl_case_conv_outbuflen()`. For general information, see [“Case Conversion” on page 2-15](#).

String Traversal

The INFORMIX-GLS library provides the following functions for string traversal of multibyte-character strings:

- `ifx_gl_mblen()`
- `ifx_gl_mbsnext()`
- `ifx_gl_mbsprev()`

For general information, see [“String Traversal” on page 2-30](#).

String Processing

The INFORMIX-GLS library provides the following functions for string processing of multibyte and wide characters.

Multibyte-Character Classification	Wide-Character Classification
<code>ifx_gl_mbscat()</code>	<code>ifx_gl_wcscat()</code>
<code>ifx_gl_mbschr()</code>	<code>ifx_gl_wcschr()</code>
<code>ifx_gl_mbscopy()</code>	<code>ifx_gl_wcscopy()</code>

Multibyte-Character Classification	Wide-Character Classification
<code>ifx_gl_mbscspn()</code>	<code>ifx_gl_wcscspn()</code>
<code>ifx_gl_mbslen()</code>	<code>ifx_gl_wcslen()</code>
<code>ifx_gl_mbsmbs()</code>	<code>ifx_gl_wcswcs()</code>
<code>ifx_gl_mbsncat()</code>	<code>ifx_gl_wcsncat()</code>
<code>ifx_gl_mbsncpy()</code>	<code>ifx_gl_wcsncpy()</code>
<code>ifx_gl_mbsntslent()</code>	<code>ifx_gl_wcsntslent()</code>
<code>ifx_gl_mbsntsbytes()</code>	None
<code>ifx_gl_mbspbrk()</code>	<code>ifx_gl_wcspbrk()</code>
<code>ifx_gl_mbsrchr()</code>	<code>ifx_gl_wcsrchr()</code>
<code>ifx_gl_mbsspn()</code>	<code>ifx_gl_wcssp()</code>

(2 of 2)

For general information, see [“String Processing” on page 2-32](#).

Character/String Comparison and Sorting

The INFORMIX-GLS library provides the following functions to obtain localized collation order for characters and strings:

- `ifx_gl_mbscoll()`
- `ifx_gl_wcscoll()`

For general information, see [“Character/String Comparison and Sorting” on page 2-35](#).

Data Formatting

The INFORMIX-GLS library provides functions to support the conversion and formatting of locale-specific strings. This section lists the data types that involve locale-specific strings and the INFORMIX-GLS functions that support them. For general information on data formatting, see [Chapter 3, “Data Formatting.”](#)

Date and Time Conversion and Formatting Functions

The INFORMIX-GLS library provides the following functions for conversion and formatting of date and date/time strings:

- **ifx_gl_convert_date()**
- **ifx_gl_convert_datetime()**
- **ifx_gl_format_date()**
- **ifx_gl_format_datetime()**

Numeric Conversion and Formatting

The INFORMIX-GLS library provides the following functions for conversion and formatting of number strings:

- **ifx_gl_convert_number()**
- **ifx_gl_format number()**

Money Conversion and Formatting

The INFORMIX-GLS library provides the following functions for conversion and formatting of money strings:

- **ifx_gl_convert_money()**
- **ifx_gl_format_money()**

Function Reference

This section describes the syntax, usage, and return values of INFORMIX-GLS functions, in alphabetical order.

ifx_gl_case_conv_outbuflen()

The **ifx_gl_case_conv_outbuflen()** function calculates an approximation of the number of bytes required to store a case-converted multibyte character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_case_conv_outbuflen(src_mbs_bytes)
    int src_mbs_bytes;
```

src_mbs_bytes is the integer number of bytes in the buffer of multibyte characters to be case converted.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_case_conv_outbuflen()** function returns one of the following values:

- Exactly the number of bytes that a buffer of case-equivalent multibyte characters will require
- A close overapproximation of this number of bytes

This buffer can contain one or more multibyte characters. This function applies to both uppercase and lowercase conversions.

Use this function to determine whether or not case conversion of multibyte characters can be performed in place. If the value that this function returns is not equal to **src_mbs_bytes**, case conversion of multibyte characters cannot be performed in place. You must allocate a separate multibyte destination buffer. However, if the value that this function returns is exactly equal to **src_mbs_bytes**, multibyte case conversion can be performed in place.

Tip: This function does not apply to wide-character case conversions.



Return Values

- >0 The number of bytes required to store multibyte characters of length *src_mbs_bytes* after they have been converted to either lowercase or uppercase characters.
- 0 The function was not successful.

Errors

None

Related Topics

See the descriptions of the following functions: **ifx_gl_toupper()** and **ifx_gl_tolower()**.

For more information, see [“Code-Set Conversion” on page 2-22](#).

ifx_gl_conv_needed()

The **ifx_gl_conv_needed()** function determines whether code-set conversion between two code sets is required.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_conv_needed(dst_codeset, src_codeset)
    char *dst_codeset;
    char *src_codeset;
```

dst_codeset is a pointer to the name of the destination (target) code set.

src_codeset is a pointer to the name of the source code set.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_conv_needed()** function determines whether characters encoded in *src_codeset* require conversion to *dst_codeset*. It is not enough to simply compare the names of the code sets because several names can identify the same code set. For example, **8859-1**, **819**, and **Latin-1** refer to the same code set.

The code sets, *src_codeset* and *dst_codeset*, can be any of the following:

- Locale names
- Code-set names
- The IFX_GL_PROC_CS macro

For more information, see [“Specifying Code-Set Names” on page 2-25](#).

Return Values

- 0 Code-set conversion is not needed between the code sets that *dst_codeset* and *src_codeset* specify.
- 1 Code-set conversion is needed between the code sets that *dst_codeset* and *src_codeset* specify.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, the function sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EINVAL	The <i>src_codeset</i> string is not a valid locale specification or code-set name, or the code-set name could not be mapped to a code-set number.
IFX_GL_EBADF	Function cannot find the code-set registry.
IFX_GL_BADFILEFORM	Bad format found in the code-set registry.

Related Topics

See the descriptions of the following functions: **ifx_gl_cv_mconv()**, **ifx_gl_cv_outbuflen()**, **ifx_gl_cv_sb2sb_table()**, and **ifx_gl_lc_errno()**.

For more information, see [“Code-Set Conversion” on page 2-22](#).

ifx_gl_convert_date()

The **ifx_gl_convert_date()** function converts a date string to its internal date representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_convert_date(date, datestr, format)
    mi_date *date;
    char *datestr;
    char *format;
```

- date* is a pointer to the variable that holds the internal date representation that **ifx_gl_convert_date()** creates from the *datestr* date string.
- datestr* is a pointer to the first character of the date string that the function converts to its internal date representation.
- format* is a pointer to the format string that determines how to interpret the *datestr* date string. For more information, see [“Format String” on page 4-16](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_convert_date()** function converts the date string that *datestr* references to its internal date representation, which the function stores in the *date* argument. The function uses the format that the *format* string specifies to scan the *datestr* date string.

The *date* argument is a pointer to a **date** (4-byte **int**) value. ♦

The *date* argument is a pointer to an **mi_date** value. ♦

E/C

DB API

Format String

If *format* is NULL, the **ifx_gl_convert_date()** function determines the format of the *datestr* date string that it scans from the environment, as follows:

- 1. If the **DBDATE** environment variable is set, the function scans *datestr* according to the order of the format elements in **DBDATE**.
- 2. If the **GL_DATE** environment variable is set, the function scans *datestr* according to the specification of **GL_DATE**.
- 3. Otherwise, the function obtains the format from the **d_fmt** subcategory of the LC_TIME category in the current GLS locale file.

If *format* is not NULL, it must point to a string that follows the rules that this section describes. To convert the *datestr* date string, **ifx_gl_convert_date()** compares each character in *datestr* with the *format* string. It takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Conversion Action Taken
One or more white-space characters	The function skips over the corresponding number of white-space characters in the <i>datestr</i> date string (unless the formatting directive begins with the minus-sign modifier), up to the first character that is not white space or until no more characters can be scanned. White-space characters are those that the blank class of the LC_CTYPE category in the current locale defines. To execute a series of formatting directives composed of %n, %t, white-space characters, or any combination, the function scans up to the first character that is not white space (which remains unscanned) or until no more characters can be scanned.
A valid formatting directive	The function performs the specified conversion on the date element in the <i>datestr</i> date string. It replaces the formatting directive with an internal representation of the date element for conversion to the <i>date</i> value. There must be white-space or other nonalphanumeric characters between any two formatting directives.
Ordinary characters	The function must find the same ordinary character in the <i>datestr</i> date string. Any mismatch generates an error. The differing and subsequent characters in <i>datestr</i> remain unscanned. You cannot include white-space characters as ordinary characters.

The formatting directive consists of the following sequence:

```
%[modifiers][flags][maximum_width][.minimum_width]type_specifier
```

Argument	Reference
modifiers	“Modified Formatting Directives” on page 4-19
flags	“Field Width” on page 4-21
maximum_width	“Field Width” on page 4-21
minimum_width	“Field Width” on page 4-21
type_specifier	“Valid Type Specifiers” on page 4-17



Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter (or letters) within a formatting directive that identifies a format for the **ifx_gl_convert_date()** function to expect when it scans a date element of a *datestr* string. These date formats are those that the LC_TIME locale-file category of the current locale might define. The **ifx_gl_convert_date()** function supports the following formatting directives to represent a date element.

Formatting Directive	Description
%a	Matches the day of the week. You can specify either the abbreviated or full weekday names, which the abday and day subcategories of LC_TIME define, respectively.
%A	Is the same as %a.
%b	Matches the month. You can specify either the abbreviated or full month names, which the abmon and mon subcategories of LC_TIME define, respectively.
%B	Is the same as %b.

Formatting Directive	Description
%C	Matches the century number [0,99]. Leading zeros are permitted but not required. If %C is used without %y , it is ignored.
%d	Matches the day of the month as a decimal number [01,31]. Leading zeros are permitted but not required.
%e	Is the same as %d .
%h	Is the same as %b .
%j	Matches the day of the year as a decimal number [001,366]. Leading zeros are permitted but not required.
%m	Matches the month as a decimal number [01,12]. Leading zeros are permitted but not required.
%n	Matches any horizontal white space that the blank class of the LC_CTYPE category defines.
%t	Matches any vertical white space that the space class of the LC_CTYPE category defines.
%u	Matches the weekday as a decimal number [1,7], with 0 representing Sunday. Leading zeros are permitted but not required.
%w	Matches the weekday as a decimal number [0,6], with 0 representing Sunday. Leading zeros are permitted but not required.
%x	Indicates use of the format that the d_fmt subcategory of LC_TIME defines.
%y	Matches the year within century as a decimal number [00,99]. Leading zeros are permitted but not required. If %y is used without %C and the month and day of month are part of the <i>datestr</i> string, the function determines the century from the DBCENTURY environment variable.
%Y	Matches the year, including the century, as a decimal number [0000,9999].
%%	Matches the % symbol.

(2 of 2)



Tip: The *ifx_gl_convert_date()* function ignores case when it matches items such as month or weekday names.

If the *format* string contains redundant formatting directives, directives that are closer to the end of the *format* string take precedence over those closer to the beginning of the *format* string.

If a formatting directive does not correspond to any of the preceding directives, the behavior of the conversion is undefined.

Modified Formatting Directives

You can modify some formatting directives with format modifiers, which follow the percent symbol (%), to indicate use of an alternative format that the LC_TIME locale-file category of the current locale might define. The **ifx_gl_convert_date()** function supports the following format modifiers.

Format Modifier	Purpose	LC_TIME Subcategory
E	Use formats that include the era-based dates.	era, era_d_fmt
O	Use alternative locale-specific digits in dates.	alt_digits

The alternative format replaces one that an unmodified formatting directive normally uses. If the alternative format does not exist for the current locale, the behavior is the same as if unmodified formatting directives were used. The **ifx_gl_convert_date()** function supports the following modified formatting directives with the **E** and **O** modifier.

Formatting Directive	Description
%EC	Matches the name of the base year (period) in the alternative representation. You can specify either the abbreviated or full name, which the era subcategory of LC_TIME defines.
%Eg	Is the same as %EC.
%Ex	Indicates use of the format that the era_d_fmt subcategory of LC_TIME category defines.
%Ey	Matches the offset from %EC (year only) in the alternative representation, which the era subcategory of LC_TIME defines.

Formatting Directive	Description
%EY	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the full alternative year representation, which the era subcategory of LC_TIME defines.
%Od	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the day of the month. Leading zeros are permitted but not required.
%Oe	Is the same as %Od .
%Om	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the month.
%Ow	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the weekday as a number (Sunday=0).
%Oy	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the value of %Ey .

(2 of 2)

The **ifx_gl_convert_date()** function needs the era base, era offset, day, and month to determine the year from era information. The function uses a default era offset of 1 if you specify era base, day, and month but not era offset.

The **ifx_gl_convert_date()** function also supports the **i** modifier in the following formatting directives to support formats that are backward compatible with earlier Informix date and time formatting.

Formatting Directive	Description
%iy	Indicates use of the Informix DBDATE format Y2. Both 98 and 1998 are interpreted as 1998 (or the century that DBCENTURY indicates).
%iY	Indicates use of the Informix DBDATE format Y4. Both 98 and 1998 are interpreted as 1998 (or the century that DBCENTURY indicates).

Field Width

You can modify some formatting directives with the following modifiers to indicate use of an alternative format that the LC_TIME locale-file category of the current locale might define:

[- | 0][*maximum_width*][.*minimum_width*]

[- | 0] indicates field justification.

If the specification begins with a minus sign (-), the function assumes that the field is left justified. In this case, the value that is being read must start with a digit and can be trailed with spaces. The field can be preceded with leading spaces unless the first character is 0. If the specification begins with 0, the function assumes that the field is right justified. Any padding to the left must use zeros. Otherwise, the function assumes that the *datestr* string is right justified.

maximum-width is a decimal number that indicates an optional maximum field width.

minimum-width indicates an optional minimum field width. The minimum field width has the format of a period (.) followed by a decimal number. The *minimum_width* decimal number represents the minimum number of characters to read. If the function reads fewer than *minimum_width* characters, it generates an error (unless you specified left justification). When you specify left justification, the function skips any trailing white space to read the required number of characters.

Locale Information

The LC_TIME category of the current locale affects the behavior of this function because it provides the locale-specific information for the scan of the *datestr* date string. For more information, see [“LC_TIME Locale-File Category” on page 3-6](#).

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurs, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_E2BIG	Operation would overflow buffer.
IFX_GL_EBADF	A formatting directive is invalid.
IFX_GL_INVALIDFMT	The <i>format</i> string is invalid.
IFX_GL_EDAYRANGE	Day number is out of bounds.
IFX_GL_EWKDAYRANGE	Weekday number is out of bounds.
IFX_GL_EYDAYRANGE	Yearday number is out of bounds.
IFX_GL_EMONTHRANGE	Month number is out of bounds.
IFX_GL_EYEARRANGE	Year number is out of bounds.
IFX_GL_EERAOFFRANGE	Era offset is out of bounds.
IFX_GL_BADDAY	Month (as a number) could not be scanned.
IFX_GL_BADWKDAY	Weekday (as a number) could not be scanned.
IFX_GL_BADYDAY	Day of year (as a number) could not be scanned.
IFX_GL_BADMONTH	Month could not be scanned.
IFX_GL_BADYEAR	Year could not be scanned.
IFX_GL_BADERANAME	Era name is invalid.
IFX_GL_BADERAOFFSET	Era offset is invalid.
IFX_GL_BADFMTMOD	Format modifier is invalid.

Value	Description
IFX_GL_BADFMTWP	Width is invalid.
IFX_GL_BADINPUT	Input string does not match <i>format</i> string.
IFX_GL_NOPOINT	Missing decimal point in input string.
IFX_GL_BADMONTHSTR	Month string could not be scanned.
IFX_GL_BADERASPEC	Could not load era from locale.

(2 of 2)

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_datetime()**, **ifx_gl_format_date()**, and **ifx_gl_lc_errno()**.

ifx_gl_convert_datetime()

The **ifx_gl_convert_datetime()** function converts a date/time string to its internal date/time representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_convert_datetime(datetime, datetimestr, format)
    mi_datetime *datetime;
    char *datetimestr;
    char *format;
```

- datetime* is a pointer to the variable that holds the internal date/time representation that **ifx_gl_convert_datetime()** creates from the *datetimestr* date/time string.
- datetimestr* is a pointer to the first character of the date/time string that the function converts to its internal date/time representation.
- format* is a pointer to the format string that determines how to interpret the *datetimestr* date/time string. For more information, see [“Format String” on page 4-25](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_convert_datetime()** function converts the date/time string that *datetimestr* references to its internal date/time representation, which the function stores in the *datetime* argument. The function uses the format that the *format* string specifies to scan the *datetimestr* date/time string.

The *datetime* argument is a pointer to a **datetime (dttime_t)** value. ♦

The *datetime* argument is a pointer to an **mi_datetime** value. ♦

E/C

DB API

Format String

If *format* is NULL, the **ifx_gl_convert_datetime()** function determines the format of the *datetimestr* date / time string that it scans from the environment, as follows:

1. If the **DBTIME** environment variable is set, the function scans *datetimestr* according to **DBTIME**.
2. If the **GL_DATETIME** environment variable is set, the function scans *datetimestr* according to the specification of **GL_DATETIME**.
3. Otherwise, the function obtains the format from the value of the **d_t_fmt** subcategory of the LC_TIME category in the current GLS locale.

If *format* is not NULL, it must point to a string that follows the rules that this section describes. To convert the *datetimestr* date / time string, **ifx_gl_convert_datetime()** compares each character in *datetimestr* with the *format* string. It takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Conversion Action Taken
One or more white-space characters	The function skips over the corresponding number of white-space characters in the <i>datetimestr</i> date / time string (unless the formatting directive begins with the minus-sign modifier), up to the first character that is not white space or until no more characters can be scanned. White-space characters are those that the blank class of the LC_CTYPE category in the current locale defines. To execute a series of formatting directives composed of %n, %t, white-space characters, or any combination, the function scans up to the first character that is not white space (which remains unscanned) or until no more characters can be scanned.
A valid formatting directive	The function performs the specified conversion on the date or time element in the <i>datetimestr</i> date / time string. It replaces the formatting directive with an internal representation of the date or time element for conversion to the <i>datetime</i> value. There must be white-space or other nonalphanumeric characters between any two formatting directives.

(1 of 2)

Contents of Format String	Conversion Action Taken
Ordinary characters	The function must find the same ordinary character in the <i>datetimestr</i> date / time string. Any mismatch generates an error. The differing and subsequent characters in <i>datetimestr</i> remain unscanned. You cannot include white-space characters as ordinary characters.

(2 of 2)

The formatting directive consists of the following sequence:

%[modifiers][flags][maximum_width][.minimum_width]type_specifier

Argument	Reference
<i>modifiers</i>	“Modified Formatting Directives” on page 4-29
<i>flags</i>	“Field Width” on page 4-32
<i>maximum_width</i>	“Field Width” on page 4-32
<i>minimum_width</i>	“Field Width” on page 4-32
<i>type_specifier</i>	“Valid Type Specifiers” on page 4-27



Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter or letters within a formatting directive that identify a format for the **ifx_gl_convert_datetime()** function to expect when it scans a date or time element of a *datetimestr* string. These date and time formats are those that the LC_TIME locale-file category of the current locale might define. The **ifx_gl_convert_datetime()** function supports the following formatting directives to represent date and time elements.

Formatting Directive	Description
%a	Matches the day of the week. You can specify either the abbreviated or full name, which the abday and day subcategories of LC_TIME define, respectively. The directive ignores case when it matches weekday names.
%A	Is the same as %a .
%b	Matches the month. You can specify either the abbreviated or full name, which the abmon and mon subcategories of LC_TIME define, respectively. The directive ignores case when it matches month names.
%B	Is the same as %b .
%c	Indicates use of the format that the d_t_fmt subcategory of the LC_TIME defines for the scan.
%C	Matches the century number [0,99]. Leading zeros are permitted but not required. If %C is used without %y , it is ignored.
%d	Matches the day of the month as a decimal number [01,31]. Leading zeros are permitted but not required.
%D	Is the same as %m / %d / %y .
%e	Is the same as %d .
%F[n]	Matches the microsecond as a decimal number [0,999999]. Leading zeros are permitted but not required. An optional precision specification can follow the %F . This <i>n</i> value must be between 1 and 5.
%h	Is the same as %b .
%H	Matches the hour (24-hour clock) as a decimal number [00,23]. Leading zeros are permitted but not required.

(1 of 3)

Formatting Directive	Description
%I	Matches the hour (12-hour clock) as a decimal number [00,12]. Leading zeros are permitted but not required. If %I is used without %p , the function assumes A.M.
%j	Matches the day of the year as a decimal number [001,366]. Leading zeros are permitted but not required.
%M	Matches the minute as a decimal number [00,59]. Leading zeros are permitted but not required.
%n	Matches any white space that the blank class of the LC_CTYPE category defines.
%p	Matches the equivalent of either A.M. or P.M. that the am_pm subcategory of LC_TIME defines.
%r	Indicates use of the format that the t_fmt_ampm subcategory of LC_TIME defines.
%R	Matches the time as %H:%M .
%S	Matches the second as a decimal number [0,61]. Leading zeros are permitted but not required.
%t	Matches any white space that the space class of the LC_CTYPE category defines.
%T	Matches the time as %H:%M:%S .
%u	Matches the weekday as a decimal number [1,7], with 0 representing Sunday. Leading zeros are permitted but not required.
%w	Matches the weekday as a decimal number [0,6], with 0 representing Sunday. Leading zeros are permitted but not required.
%x	Indicates use of the format that the d_fmt subcategory of LC_TIME defines.
%X	Indicates use of the format that the t_fmt subcategory of LC_TIME defines.

(2 of 3)


**Formatting
Directive**
Description

%y	Matches a year within the century as a decimal number [00,99]. Leading zeros are permitted but not required. If %y is used without %C and the month and day of month are part of the <i>datetimestr</i> string, the function uses the value of the DBCENTURY environment variable to determine the century.
%Y	Matches a year that includes the century as a decimal number [0000,9999].
%%	Matches the % character.

(3 of 3)

Tip: The *ifx_gl_convert_datetime()* function ignores case when it matches items such as month or weekday names.

If the *format* string contains redundant formatting directives, directives that are closer to the end of the *format* string take precedence over those closer to the beginning of the *format* string.

If a formatting directive does not correspond to any of the preceding directives, the behavior of the conversion is undefined.

Modified Formatting Directives

You can modify some formatting directives with format modifiers, which follow the percent symbol (%), to indicate use of an alternative format that the LC_TIME locale-file category of the current locale might define. The *ifx_gl_convert_datetime()* function supports the following format modifiers.

Format Modifier	Purpose	LC_TIME Subcategory
E	Use formats that include the era-based dates and times.	era, era_d_fmt, era_t_fmt, era_d_t_fmt
O	Use alternative locale-specific digits in dates and times.	alt_digits

The alternative format replaces one that an unmodified formatting directive normally uses. If the alternative format does not exist for the current locale, the behavior is the same as if unmodified formatting directives were used. The **ifx_gl_convert_datetime()** function supports the following modified formatting directives with the **E** and **O** modifiers.

Formatting Directive	Description
%Ec	Indicates use of the format that the era_d_t_fmt subcategory of LC_TIME defines.
%EC	Matches the name of the base year (period). You can specify either the abbreviated or full name, which the era subcategory of LC_TIME defines.
%Eg	Is the same as %EC .
%Ex	Indicates use of the format that the era_d_fmt subcategory of LC_TIME defines.
%EX	Indicates use of the format that the era_t_fmt subcategory of LC_TIME defines.
%Ey	Matches the year offset from %EC in the alternative representation, which the era subcategory of LC_TIME defines.
%EY	Matches the full alternative year representation that the era subcategory of LC_TIME defines.
%Od	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the day of the month. Leading zeros are permitted but not required.
%Oe	Is the same as %Od .
%OH	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the hour (24-hour clock).
%OI	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the hour (12-hour clock).
%Om	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the month.

(1 of 2)

Formatting Directive	Description
%OM	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the minutes.
%OS	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the seconds.
%Ow	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the weekday as a number (Sunday=0).
%Oy	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to match the value of %Ey .

(2 of 2)

The **ifx_gl_convert_datetime()** function needs the era base, era offset, day, and month to determine the year from era information. The function uses a default era offset of 1 if you specify era base, day, and month but not era offset.

The **ifx_gl_convert_datetime()** function also supports the **i** modifier in the following formatting directives to support formats that are backward compatible with earlier Informix date and time formatting.

Formatting Directive	Description
%iF[n]	Indicates use of the Informix DBTIME format %F . An optional precision specification, <i>n</i> , can follow F . This <i>n</i> value must be between 1 and 5.
%iy	Indicates use of the Informix DBDATE format Y2 . Both 98 and 1998 are interpreted as 1998 (or the century that DBCENTURY indicates).
%iY	Indicates use of the Informix DBDATE format Y4 . Both 98 and 1998 are interpreted as 1998 (or the century that DBCENTURY indicates).

Field Width

You can specify an optional field width in a formatting directive. This information follows the initial percent symbol (%) of the formatting directive and has the following format:

[- | 0][*maximum_width*][. *minimum_width*]

[- | 0] indicates field justification.

If the specification begins with a minus sign (-), the function assumes that the field is left justified. In this case, the value being read must start with a digit and can have trailing spaces. The field can be preceded with leading spaces unless the first character in the field is 0.

If the first character is 0, the field is assumed to be right justified. Any padding to the left must use zeros.

Otherwise, the function assumes that the *datetimestr* string is right justified.

maximum_width is a decimal value that specifies the maximum number of characters to read from the *datetimestr* date / time string.

minimum_width indicates an optional minimum field width. The minimum field width has the format of a period (.) followed by a decimal number. The *minimum_width* decimal number represents the minimum number of characters to read. If the function reads fewer than *minimum_width* characters, it generates an error (unless you specified left justification). When you specify left justification, the function skips any trailing white space to read the required number of characters.



Tip: The *ifx_gl_convert_datetime()* function ignores any field-width specification for any nonnumeric or compound formats.

For the **%Fn** format, *n* overrides the minimum field width. If *n* is greater than the maximum field width, the maximum field width is increased to *n*. If *n* is not between 1 and 5, an error is returned.

Locale Information

The LC_TIME category of the current locale affects the behavior of this function because it provides the locale-specific information for the scan of the *datetimestr* date / time string. For more information, see [“LC_TIME Locale-File Category” on page 3-6](#).

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EBADF	A formatting directive is invalid.
IFX_GL_EFRACRANGE	Fraction of second is out of bounds.
IFX_GL_ESECONDRANGE	Second is out of bounds.
IFX_GL_EMINUTERANGE	Minute is out of bounds.
IFX_GL_EHOURRANGE	Hour is out of bounds.
IFX_GL_EDAYRANGE	Day number is out of bounds.
IFX_GL_EWKDAYRANGE	Weekday number is out of bounds.
IFX_GL_EYDAYRANGE	Yearday number is out of bounds.
IFX_GL_EMONTHRANGE	Month number is out of bounds.
IFX_GL_EYEARRANGE	Year number is out of bounds.
IFX_GL_EERAOFFRANGE	Era offset is out of bounds.
IFX_GL_BADFRAC	Fraction could not be scanned.

(1 of 2)

Value	Description
IFX_GL_BADSECOND	Second could not be scanned.
IFX_GL_BADMINUTE	Minute could not be scanned.
IFX_GL_BADHOUR	Hour could not be scanned.
IFX_GL_BADDAY	Month (as a number) could not be scanned.
IFX_GL_BADWKDAY	Weekday (as a number) could not be scanned.
IFX_GL_BADYDAY	Day of year (as a number) could not be scanned.
IFX_GL_BADMONTH	Month could not be scanned.
IFX_GL_BADYEAR	Year could not be scanned.
IFX_GL_BADERANAME	Era name was not found.
IFX_GL_BADERAOFFSET	Era offset could not be scanned.

(2 of 2)

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_date()**, **ifx_gl_format_datetime()**, and **ifx_gl_lc_errno()**.

ifx_gl_convert_money()

The **ifx_gl_convert_money()** function converts a money string to its internal money representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_convert_money(money, monstr, format)
    mi_money *money;
    char *monstr;
    char *format;
```

- money* is a pointer to the variable that holds the internal money representation that **ifx_gl_convert_money()** creates from the *monstr* money string.
- monstr* is a pointer to the first character of the money string that the function converts to its internal money representation.
- format* is a pointer to the format string that determines how to interpret the *monstr* money string. For more information, see [“Format String” on page 4-36](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_convert_money()** function converts the money string that *monstr* references to its internal money representation, which the function stores in the *money* argument. The function uses the format that the *format* string specifies to scan the *monstr* money string.

The *money* argument is a pointer to a **decimal (dec_t)** value. ♦

The *money* argument is a pointer to an **mi_money** value. ♦

E/C

DB API

Format String

If *format* is NULL, the **ifx_gl_convert_money()** function determines the format of the *monstr* money string that it scans from the environment, as follows:

- 1. If the **DBMONEY** environment variable is set, the function scans *monstr*, as **DBMONEY** specifies.
- 2. Otherwise, the function uses the formats in the LC_MONETARY category of the current GLS locale to scan the *money* value.

If *format* is not NULL, it must point to a string that follows the rules that this section describes. To convert the *monstr* money string, the function **ifx_gl_convert_money()** compares each character in *monstr* with the *format* string. It takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Conversion Action Taken
One or more white-space characters	The function skips over the corresponding number of white-space characters in the <i>monstr</i> money string (unless the formatting directive begins with the minus-sign modifier), up to the first character that is not white space or until no more characters can be scanned. White-space characters are those that the blank class of the LC_CTYPE category in the current locale defines.
A valid formatting directive	The function performs the specified conversion on the monetary element in the <i>monstr</i> money string. It replaces the formatting directive with an internal representation of the monetary element for conversion to the <i>money</i> value. Only one formatting directive is allowed in the <i>format</i> string.
Ordinary characters	The function must find the same ordinary character in the <i>monstr</i> money string. Any mismatch generates an error. The differing and subsequent characters in <i>monstr</i> remain unscanned. You cannot include white-space characters as ordinary characters.

The formatting directive consists of the following sequence:

`%[ flags ][ maximum_width ] type_specifier`

Argument	Reference
<i>flags</i>	“Field Width” on page 4-38
<i>maximum_width</i>	“Field Width” on page 4-38
<i>type_specifier</i>	“Valid Type Specifiers” on page 4-37



Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter or letters within a formatting directive that identify a format for the **ifx_gl_convert_money()** function to expect when it scans a monetary element of a *monstr* string. The LC_MONETARY locale-file category of the current locale might define these monetary formats. The **ifx_gl_convert_money()** function supports the following formatting directives to represent a monetary element.

Formatting Directive	Description
%i	Matches the international monetary format (which uses the int_curr_symbol subcategory of the LC_MONETARY category in the current locale) for the <i>monstr</i> argument. For example, in the default locale, the international monetary format for 1,234.56 is the “USD 1,234.56” string. Declare the corresponding <i>money</i> value as (double *).
%n	Matches the national currency format (which uses the currency_symbol of the LC_MONETARY category in the current locale) for the <i>monstr</i> argument. For example, in the default locale, the national monetary format for 1,234.56 is the “\$1,234.56” string. Declare the corresponding <i>money</i> value as (double *).
%%	Matches a % character. No corresponding argument is needed.

If a formatting directive does not correspond to any of the preceding directives, the behavior of the conversion is undefined.

Field Width

You can specify an optional field width in a formatting directive. This information follows the initial percent symbol (%) of the formatting directive and has the following format:

`[ ' ] [ - ] [ maximum_width ]`

`[ ' ]` indicates grouping rules.

If the specification begins with a single quote ('), the field is expected to follow the grouping rules in the LC_MONETARY category of the current locale.

`[ - ]` indicates field justification.

If the specification begins with a minus sign (-), the function expects the first character of the *monstr* money string to be the first character of the value to be converted. The function does not skip white-space characters in *monstr* but reports them as an error.

If you omit the minus sign (-), the function assumes that the *monstr* string is right justified, and it skips any initial white-space characters.

maximum_width is a decimal value that specifies the maximum number of characters to read from the *monstr* money string.

Locale Information

The LC_MONETARY category of the current locale affects the behavior of this function because it provides the locale-specific information for the scan of the *monstr* money string. For more information, see [“LC_MONETARY Locale-File Category” on page 3-5](#).

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_INVALIDFMT	The <i>format</i> string is invalid.

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_numeric()**, **ifx_gl_format_money()**, and **ifx_gl_lc_errno()**.

ifx_gl_convert_number()

The **ifx_gl_convert_number()** function converts a number string to its internal decimal representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_convert_number(number, numstr, format)
    mi_decimal *number;
    char *numstr;
    char *format;
```

- number* is a pointer to the variable that holds the internal decimal representation that **ifx_gl_convert_number()** creates from the *numstr* number string.
- numstr* is a pointer to the first character of the number string that the function converts to its internal decimal representation.
- format* is a pointer to the format string that determines how to interpret the *numstr* number string. For more information, see [“Format String” on page 4-41](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_convert_number()** function converts the number string that *numstr* references to its internal decimal representation, which the function stores in the *number* argument. The function uses the format that the *format* string specifies to scan the *numstr* number string.

The *number* argument is a pointer to a **decimal (dec_t)** value. ♦

The *number* argument is a pointer to an **mi_decimal** value. ♦

E/C

DB API

Format String

If *format* is NULL, the function uses the formats in the LC_NUMERIC category of the current GLS locale file to scan the *numstr* number string. If *format* is not NULL, it must point to a string that follows the rules that this section describes.

To convert the *numstr* number string, **ifx_gl_convert_number()** compares each character in *numstr* with the *format* string. It takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Conversion Action Taken
One or more white-space characters	The function skips the corresponding number of white-space characters in the <i>numstr</i> number string (unless the formatting directive begins with the minus-sign modifier), up to the first character that is not white space or until no more characters can be scanned. White-space characters are those that the blank class of the LC_CTYPE category in the current locale defines.
A valid formatting directive	The function performs the specified conversion on the numeric element in the <i>numstr</i> numeric string. It replaces the formatting directive with an internal representation of the numeric element for conversion to the <i>number</i> value. Only one formatting directive is allowed in the <i>format</i> string.
Ordinary characters	The function must find the same ordinary character in the <i>numstr</i> number string. Any mismatch generates an error. The differing and subsequent characters in <i>numstr</i> remain unscanned.



The formatting directive consists of the following sequence:

`%[ flags ][ maximum_width[ .minimum_width ] ] type_specifier`

Argument	Reference
<i>flags</i>	“Field Width” on page 4-43
<i>maximum_width</i>	“Field Width” on page 4-43
<i>minimum_width</i>	“Field Width” on page 4-43
<i>type_specifier</i>	“Valid Type Specifiers” on page 4-42

Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter or letters within a formatting directive that identify a format for the **ifx_gl_convert_number()** function to expect when it scans a numeric element of a *numstr* string. These numeric formats are those that the LC_NUMERIC locale-file category of the current locale might define. The **ifx_gl_convert_number()** function supports the following formatting directives to represent a numeric element.

Formatting Directive	Description
%b	Matches a binary integer.
%d	Matches a decimal integer.
%e	Matches a floating-point number.
%E	Is the same as %e .
%f	Is the same as %e .
%g	Is the same as %e .
%G	Is the same as %e .
%i	Is the same as %d .

Formatting Directive	Description
%o	Matches an octal integer.
%q	Matches a base-4 integer.
%u	Matches an unsigned decimal integer.
%x	Matches a hexadecimal integer.
%X	Is the same as %x .

(2 of 2)

If a formatting directive does not correspond to any of the preceding directives, the behavior of the conversion is undefined.

Field Width

You can specify an optional field width in a formatting directive. This information follows the initial percent symbol (%) of the formatting directive and has the following format:

```
[ - ][ maximum_width ][ . minimum_width ]
```

[-] indicates field justification.

If the specification begins with a minus sign (-), the function expects the first character of the *numstr* number string to be the first character of the value to be converted. The function does not skip white-space characters in *numstr* but reports them as an error.

If you omit the minus sign (-), the function assumes that the *numstr* string is right justified, and it skips any initial white-space characters.

maximum_width is a decimal value that specifies the maximum number of characters to read from the *numstr* number string.

minimum_width indicates an optional minimum field width. The minimum field width has the format of a period (.) followed by a decimal number. The *minimum_width* decimal number represents the minimum number of characters to read. If the function reads fewer than *minimum_width* characters, it generates an error (unless you specified left justification). When you specify left justification, the function skips any trailing white space to read the required number of characters.

Locale Information

The LC_NUMERIC category of the current locale affects the behavior of this function because it provides locale-specific information for the scan of the *numstr* number string. For more information, see “[LC_NUMERIC Locale-File Category](#)” on page 3-4.

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_INVALIDFMT	The <i>format</i> string is invalid.
IFX_GL_PARAMERR	The type specifier in the formatting directive is invalid.

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_money()**, **ifx_gl_format_number()**, and **ifx_gl_lc_errno()**.

ifx_gl_cv_mconv()

The **ifx_gl_cv_mconv()** function converts characters from one code set to another.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_cv_mconv(state, dst, dst_bytesleft, dst_codeset, src, src_bytesleft,
    src_codeset)
    conv_state_t *state;
    gl_mchar_t **dst;
    int *dst_bytesleft;
    char *dst_codeset;
    gl_mchar_t **src;
    int *src_bytesleft;
    char *src_codeset;
```

<i>state</i>	points to a conv_state_t structure. For more information, see “Converting Fragmented Strings” on page 4-47 .
<i>dst</i>	points to a pointer to the first character in the destination buffer. If <i>dst</i> is NULL, the function updates <i>*src</i> , <i>*src_bytesleft</i> , and <i>*dst_bytesleft</i> , but the converted data is not written to <i>*dst</i> .
<i>dst_bytesleft</i>	points to the maximum number of bytes to write to <i>*dst</i> .
<i>dst_codeset</i>	is a pointer to the name of the destination (target) code set.
<i>src</i>	points to a pointer to the first character in the source buffer.
<i>src_bytesleft</i>	points to the number of bytes in <i>*src</i> to convert. If <i>src_bytesleft</i> is NULL, the function converts data until it reaches a null character in the source buffer.
<i>src_codeset</i>	is a pointer to the name of the source code set.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_cv_mconv()** function converts the string of multibyte characters in **src* to the same characters, but encoded in another code set, and stores the result in the buffer that **dst* references. The **ifx_gl_cv_mconv()** function updates the following arguments:

- The function updates *src* to point to the byte that follows the last source character successfully converted.
- The function updates the integer that *src_bytesleft* references to the number of bytes in **src* that have not been converted. After a successful conversion, *src_bytesleft* points to 0.
- The function updates *dst* to point to the first character of the code-set-converted text.
- The function updates the integer that *dst_bytesleft* references to the number of bytes still available in **dst*.

Specifying Code Sets

The code sets, *src_codeset* and *dst_codeset*, can be any of the following items:

- Locale names
- Code-set names
- The IFX_GL_PROC_CS macro

For more information, see [“Specifying Code-Set Names” on page 2-25](#).

Calculating the Size of the Destination Buffer

The number of bytes written to **dst* might be more or fewer than the number of bytes read from **src*. You can determine the number of bytes that will be written to **dst* in one of the following ways:

- The function **ifx_gl_cv_outbuflen()** calculates an estimate based on the **src_bytesleft* value.
- The following expression is the maximum number of bytes that will be written to **dst* for any value of **src* in any locale:

$$(*srcbytesleft) * IFX_GL_MB\_MAX$$

Of these options, the last method is the fastest. The **ifx_gl_cv_outbuflen()** function is the slowest but the most precise.

Converting Fragmented Strings

The *state* argument is a pointer to a **conv_state_t** structure. You must allocate a **conv_state_t** structure and set the **first_frag** and **last_frag** fields of the **conv_state_t** structure to tell the **ifx_gl_cv_mconv()** function whether the string to be converted is a fragment.

The following table lists the different fragments of a string and the corresponding values to which you must set these two **conv_state_t** fields.

String Fragment	Value of first_frag Field	Value of last_frag Field
String is the first of <i>n</i> fragments.	1	0
String is one of the 2nd, ..., <i>n</i> th-1 fragments.	0	0
String is the last (<i>n</i> th) fragment.	0	1
String is <i>not</i> fragmented; it is a complete string.	1	1



Important: The **conv_state_t** structure contains other fields that are for internal use only. Informix does not guarantee that these other internal fields of **conv_state_t** will not change in future releases. Therefore, to create portable code, set only the **first_frag** and **last_frag** fields of the **conv_state_t** structure.

For more information on the use of the **conv_state_t** structure, see [“Preserving State Information” on page 2-26](#).

Return Values

- 0 The code-set conversion was successful, and the **src*, **src_bytesleft*, **dst*, and **dst_bytesleft* arguments have been updated. If the entire source buffer is converted, the **srcbytesleft* value is 0.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section. If the source conversion is stopped due to an error, the **srcbytesleft* value is greater than 0.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_FILEERR	Retrieving the conversion information for the specified code sets failed. This failure might be due to invalid code-set names, a missing registry file, a missing code-set-conversion object file or one with an incorrect format, or insufficient memory for the code-set-conversion object.
IFX_GL_EILSEQ	The <i>*src</i> value contains an invalid character. Conversion stops after both the last successfully converted source and destination character.
IFX_GL_EINVAL	The function cannot determine whether the last character of <i>*src</i> is a valid character or the end of shift sequence because it would need to read more than <i>*src_bytesleft</i> bytes from <i>*src</i> . Conversion stops after the last successfully converted source and destination character. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .
IFX_GL_E2BIG	Not enough space is available in the destination buffer. Conversion stops after both the last successfully converted source and destination character.

Related Topics

See the descriptions of the following functions: **ifx_gl_conv_needed()**, **ifx_gl_cv_outbuflen()**, **ifx_gl_cv_sb2sb_table()**, and **ifx_gl_lc_errno()**.

For more information, see [“Code-Set Conversion” on page 2-22](#) and [“Fragmenting Multibyte Strings” on page 2-44](#).

ifx_gl_cv_outbuflen()

The **ifx_gl_cv_outbuflen()** function calculates an approximation of the number of bytes required to store a case-converted multibyte character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_cv_outbuflen(dst_codeset, src_codeset, src_bytes)
    char *dst_codeset;
    char *src_codeset;
    int src_bytes;
```

- dst_codeset* is a pointer to the name of the destination (target) code set.
- src_codeset* is a pointer to the name of the source code set.
- src_bytes* is the number of bytes in the buffer of multibyte characters to be code-set converted.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_cv_outbuflen()** function returns one of the following values:

- Exactly the number of bytes that a buffer of code-set-converted multibyte characters will require
- A close overapproximation of this number of bytes

Use this function to determine whether or not code-set conversion of multibyte characters can be performed in place. If the value that this function that returns is not equal to *src_bytes*, code-set conversion of multibyte characters cannot be performed in place. You must allocate a separate multibyte destination buffer. However, if the value that this function returns is exactly equal to *src_bytes*, you can perform multibyte code-set conversion in place.

The code sets, *src_codeset* and *dst_codeset*, can be any of the following:

- Locale names
- Code-set names
- The IFX_GL_PROC_CS macro

For more information, see [“Specifying Code-Set Names” on page 2-25](#).

Return Values

- >=0 The number of bytes required to store multibyte characters of length *src_bytes* after they have been code-set converted.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_FILEERR	Retrieval of the conversion information for the specified code sets failed. This failure might be due to invalid code-set names, a missing registry file, a missing code-set-conversion object file or one with an incorrect format, or insufficient memory for the code-set-conversion object.

Related Topics

See the descriptions of the following functions: **ifx_gl_conv_needed()**, **ifx_gl_cv_mconv()**, **ifx_gl_cv_sb2sb_table()**, and **ifx_gl_lc_errno()**.

For more information, see [“Code-Set Conversion” on page 2-22](#).

ifx_gl_cv_sb2sb_table()

The **ifx_gl_cv_sb2sb_table()** function returns the single-byte conversion table from the source code set to the destination code set.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_cv_sb2sb_table(dst_codeset, src_codeset, array)
    char *dst_codeset;
    char *src_codeset;
    unsigned char **array;
```

- dst_codeset* is a pointer to the name of the destination (target) code set.
- src_codeset* is a pointer to the name of the source code set.
- array* is the number of bytes in the buffer of multibyte characters to be code-set converted.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

If the code-set conversion from *src_codeset* to *dst_codeset* converts from one single-byte code set to another single-byte code set (where no substitution conversions occur), the **ifx_gl_cv_sb2sb_table()** function sets *array* to an array of 256 unsigned **char** values that represent the conversion. If the code-set conversion is *not* of this form, the function sets *array* to NULL.

The code sets, *src_codeset* and *dst_codeset*, can be any of the following items:

- Locale names
- Code-set names
- The IFX_GL_PROC_CS macro

For more information, see [“Specifying Code-Set Names” on page 2-25](#).

Return Values

- 0 The function was successful, and the *array* argument points to the conversion table.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_FILEERR	Retrieval of the conversion information for the specified code sets failed. This failure might be due to invalid code-set names, a missing registry file, a missing code-set-conversion object file or one with an incorrect format, or insufficient memory for the code-set-conversion object.

Related Topics

See the descriptions of the following functions: **ifx_gl_conv_needed()**, **ifx_gl_cv_mconv()**, **ifx_gl_cv_outbuflen()**, and **ifx_gl_lc_errno()**.

For more information, see [“Code-Set Conversion” on page 2-22](#).

ifx_gl_format_date()

The **ifx_gl_format_date()** function formats an internal date value to a date string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_format_date(datestr, datebytes, date, format)
    char *datestr;
    int datebytes;
    mi_date *date;
    char *format;
```

- datestr* is a pointer to the first character in the date string that the function formats from its internal *date* representation.
- datebytes* is the size of the *datestr* buffer for the date string. This size is the maximum size that the formatted string can reach.
- date* is a pointer to the variable that holds the internal representation that **ifx_gl_format_date()** formats to create the *datestr* date string.
- format* is a pointer to the format string that determines how to format the *datestr* date string. For more information, see [“Format String” on page 4-55](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_format_date()** function uses the format that the *format* string specifies to format the internal date representation in *date* as a date string. It stores the resulting date string in a buffer that *datestr* references.

The *date* argument is a pointer to a **date** (4-byte **int**) value. ♦

The *date* argument is a pointer to an **mi_date** value. ♦

E/C

DB API

Format String

If *format* is NULL, the **ifx_gl_format_date()** function determines the format for the *datestr* date string that it creates from the environment, as follows:

- 1. If the **DBDATE** environment variable is set, the function formats *datestr* according to the order of the format elements in **DBDATE**.
- 2. If the **GL_DATE** environment variable is set, the function formats *datestr* according to the specification of **GL_DATE**.
- 3. Otherwise, the function obtains the formats from the **d_fmt** subcategory of the LC_TIME category in the current GLS locale.

If *format* is not NULL, it must point to a string that follows the rules that this section describes. To format the *datestr* date string, **ifx_gl_format_date()** takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Formatting Action
Valid formatting directives	The function performs the specified formatting of the data in the <i>date</i> argument. It replaces the formatting directive with a string representation of a date element for the <i>datestr</i> date string.
Ordinary characters	The function copies the ordinary character unchanged to the <i>datestr</i> date string. Ordinary characters include the null terminator and white-space characters.

The formatting directive consists of the following sequence:

%[modifiers][flag][width][.precision]type_specifier

Argument	Reference
<i>modifiers</i>	“Modified Formatting Directives” on page 4-58
<i>flag</i>	“Field Width and Precision” on page 4-60



Argument	Reference
<i>width</i>	“Field Width and Precision” on page 4-60
<i>precision</i>	“Field Width and Precision” on page 4-60
<i>type_specifier</i>	“Valid Type Specifiers” on page 4-56

(2 of 2)

Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter or letters within a formatting directive that identify a format for the **ifx_gl_format_date()** function to use when it creates a date element of a *datestr* string. These date formats are those that the LC_TIME locale-file category of the current locale might define. The **ifx_gl_format_date()** function supports the following formatting directives to represent a date element.

Formatting Directive	Description
%a	Replaced by the abbreviated weekday name that the abday subcategory of LC_TIME defines.
%A	Replaced by the full weekday name that the day subcategory of LC_TIME defines.
%b	Replaced by the abbreviated month name that the abmon subcategory of LC_TIME defines.
%B	Replaced by the full month name that the mon subcategory of LC_TIME defines.
%C	Replaced by the century number (the year divided by 100 and truncated to an integer as a decimal number [00-00]).
%d	Replaced by the day of the month as a decimal number [01,31].
%D	Is the same as %m / %d / %y .

(1 of 2)

Formatting Directive	Description
%e	Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space.
%h	Is the same as %b .
%j	Replaced by the day of the year as a decimal number [001,366].
%m	Replaced by the month as a decimal number [01,12].
%n	Replaced by a newline character.
%t	Replaced by a tab character.
%u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday.
%w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.
%x	Indicates use of the format that the d_fmt subcategory of LC_TIME defines.
%y	Replaced by year without century as a decimal number [00,99].
%Y	Replaced by year with century as a decimal number [00,99].
%%	Replaced by the % symbol.

(2 of 2)

If a formatting directive does not correspond to any of the preceding directives, the behavior of the formatting is undefined.

Modified Formatting Directives

You can modify some formatting directives with format modifiers, which follow the percent symbol (%), to indicate use of an alternative format that the LC_TIME locale-file category of the current locale might define. The **ifx_gl_format_date()** function supports the following format modifiers.

Format Modifier	Purpose	LC_TIME Subcategory
E	Use formats that include the era-based dates.	era, era_d_fmt
O	Use alternative locale-specific digits in dates.	alt_digits

The alternative format replaces one that an unmodified formatting directive normally uses. If the alternative format does not exist for the current locale, the behavior is the same as if unmodified formatting directives were used. The **ifx_gl_format_date()** function supports the following modified formatting directives with the **E** and **O** modifiers.

Formatting Directive	Description
%EC	Replaced by the name of the base year (period) that the era subcategory of LC_TIME defines.
%Eg	Replaced by the abbreviated name of the base year (period) that the era subcategory of LC_TIME defines.
%Ex	The value of the era_d_fmt subcategory of the LC_TIME category for the current locale is temporarily used as the format string.
%Ey	Replaced by the offset from %EC (year only) that the era subcategory of LC_TIME defines.
%EY	Replaced by the full alternative year representation that the era subcategory of LC_TIME defines.
%Oe	Replaced by the day of the month, using the alternative digits that the alt_digits subcategory of LC_TIME defines. This formatting directive fills as needed with leading spaces.

(1 of 2)

Formatting Directive	Description
%Om	Replaced by the month, using the alternative digits that the alt_digits subcategory of LC_TIME defines.
%Ou	Replaced by the weekday as a number, using the alternative digits that the alt_digits subcategory of LC_TIME defines (Monday=1).
%Oy	Replaced by the value of %Ey , using the alternative digits that the alt_digits subcategory of LC_TIME defines.

(2 of 2)

The **ifx_gl_format_date()** function also supports the **i** modifier in the following formatting directives to support formats that are backward compatible with earlier Informix date and time formatting.

Formatting Directive	Description
%iy	Replaced by the Informix DBDATE format Y2. This format prints the two-digit year offset.
%iY	Replaced by the Informix DBDATE format Y4. This format prints the full four-digit year.

Field Width and Precision

You can also specify an optional field width and precision in a formatting directive. This information precedes the type specifier in the formatting directive and has the following format:

```
[ - | 0 ][width][.precision]
```

[- | 0] indicates field justification.

If the specification begins with a minus sign (-), the function left-justifies the field and pads it with spaces on the right.

If the value begins with a zero (0), the function right-justifies the field and pads it with zeros on the left.

Otherwise, the function right-justifies the field and pads it with spaces on the left.

width is a decimal value that specifies a minimum field width for the formatted *datestr* value.

precision has the format of a period (.) followed by a decimal value. The field *width* specifier can optionally be followed by a *precision* directive. For more information on the purpose of *precision*, see the following table.

The values of *width* and *precision* affect each element of the directive. For example, **%6.4D** is interpreted as "%6.4m / %6.4d / %6.4y".

The *precision* value has the following effects on the different formatting directives.

Formatting Directives	Effect of Precision
%C, %d, %e, %Ey, %iy, %iY, %j, %m, %u, %w, %y, %Y	The value of <i>precision</i> specifies the minimum number of digits to appear. If a directive supplies fewer digits than <i>precision</i> specifies, the function pads the value with leading zeros. The %d, %Ey, %iy, %m, %u, %w, and %y directives have a default precision of 2. The %j directive has a default precision of 3. The %Y and %iY directives have a default precision of 4.
%a, %A, %b, %B, %EC, %Eg, %h	The value of <i>precision</i> specifies the maximum number of characters to be used. If a value to be formatted has more characters than the <i>precision</i> specifies, the function truncates the result on the right.
%Ex, %EY, %n, %t, %x, %%	The values of <i>width</i> and <i>precision</i> do not affect these directives.
Directives modified with O (alternative digits)	The field <i>width</i> relates to display width rather than actual number of digits; <i>precision</i> is still the minimum number of digits printed.

Locale Information

The LC_TIME category of the current locale affects the behavior of this function because it provides the locale-specific information for the formatting of the *date* value. For more information, see “[LC_TIME Locale-File Category](#)” on page 3-6.

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_E2BIG	Formatting stopped due to lack of space in the buffer.
IFX_GL_EBADF	The formatting directive is invalid.
IFX_GL_EDAYRANGE	Day number is out of bounds.
IFX_GL_EWKDAYRANGE	Weekday number is out of bounds.
IFX_GL_EYDAYRANGE	Year day number is out of bounds.
IFX_GLEMONTHRANGE	Month number is out of bounds.
IFX_GL_EYEARRANGE	Year number is out of bounds.
IFX_GL_EERAOFFRANGE	Era offset is out of bounds.
IFX_GL_BADDAY	Month (as a number) could not be scanned.
IFX_GL_BADWKDAY	Weekday (as a number) could not be scanned.
IFX_GL_BADYDAY	Day of year (as a number) could not be scanned.
IFX_GL_BADMONTH	Month could not be scanned.
IFX_GL_BADYEAR	Year could not be scanned.
IFX_GL_BADERANAME	Era name was not found.
IFX_GL_BADERAOFFSET	Era offset could not be scanned.

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_date()**, **ifx_gl_format_datetime()**, and **ifx_gl_lc_errno()**.

ifx_gl_format_datetime()

The **ifx_gl_format_datetime()** function formats an internal date/time value to a date/time string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_format_datetime(datetimestr, datetimebytes, datetime, format)
    char *datetimestr;
    int datetimebytes;
    mi_datetime *datetime;
    char *format;
```

<i>datetimestr</i>	is a pointer to the first character in the date/time string that the function formats from its internal <i>datetime</i> representation.
<i>datetimebytes</i>	is the size of the <i>datetimestr</i> buffer for the date/time string. This size is the maximum size that the formatted string can reach.
<i>datetime</i>	is a pointer to the variable that holds the internal representation that ifx_gl_format_datetime() formats to create the <i>datetimestr</i> date/time string.
<i>format</i>	is a pointer to the format string that determines how to format the <i>datetimestr</i> date/time string. For more information, see “Format String” on page 4-64 .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_format_datetime()** function uses the format that the *format* string specifies to format the internal date/time representation in *datetime* as a date/time string. It stores the resulting date/time string in a buffer that *datetimestr* references.

The *datetime* argument is a pointer to a **datetime** (**dtime_t**) value. ♦

The *datetime* argument is a pointer to an **mi_datetime** value. ♦

Format String

If *format* is NULL, the **ifx_gl_format_datetime()** function determines the format for the *datetimestr* date/time string that it creates from the environment, as follows:

- 1. If the **DBTIME** environment variable is set, the function formats *datetimestr* according to **DBTIME**.
- 2. If the **GL_DATETIME** environment variable is set, the function formats *datetimestr* according to **GL_DATETIME**.
- 3. Otherwise, the function obtains the formats from the **d_t_fmt** subcategory of the LC_TIME category in the current GLS locale.

If *format* is not NULL, it must point to a string that follows the rules that this section describes. To format the *datetimestr* date/time string, **ifx_gl_format_datetime()** takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Formatting Action Taken
Valid formatting directives	The function performs the specified formatting of the data in the <i>datetime</i> argument. It replaces the formatting directive with a string representation of a date or time element for the <i>datetimestr</i> date/time string.
Ordinary characters	The function copies the ordinary character unchanged to the <i>datetimestr</i> date/time string. Ordinary characters include the null terminator and white-space characters.

The formatting directive consists of the following sequence:

```
%[modifiers][flag][width][.precision]type_specifier
```

Argument	Reference
<i>modifiers</i>	“Modified Formatting Directives” on page 4-67
<i>flags</i>	“Field Width and Precision” on page 4-70
<i>width</i>	“Field Width and Precision” on page 4-70
<i>precision</i>	“Field Width and Precision” on page 4-70
<i>type_specifier</i>	“Valid Type Specifiers” on page 4-65



Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter or letters within a formatting directive that identify a format for the **ifx_gl_format_datetime()** function to use when it creates a date or time element of a *datetimestr* string. These date and time formats are those that the LC_TIME locale-file category of the current locale might define. The **ifx_gl_format_datetime()** function supports the following formatting directives to represent a date or time element that the LC_TIME locale-file category of the current locale might define.

Formatting Directive	Description
%a	Replaced by the abbreviated weekday name that the abday subcategory of LC_TIME defines.
%A	Replaced by the full weekday name that the day subcategory of LC_TIME defines.
%b	Replaced by the abbreviated month name that the abmon subcategory of LC_TIME defines.
%B	Replaced by the full month name that the mon subcategory of LC_TIME defines.

(1 of 3)

Formatting Directive	Description
%c	Indicates use of the format that the d_t_fmt subcategory of LC_TIME defines.
%C	Replaced by the century number (the year divided by 100 and truncated to an integer as a decimal number [00-00]).
%d	Replaced by the day of the month as a decimal number [01,31].
%D	Is the same as %m / %d / %y .
%e	Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space.
%F[n]	Replaced by the microsecond as a decimal number. An optional precision specification, <i>n</i> , can follow F . This <i>n</i> value must be between 1 and 5.
%h	Is the same as %b .
%H	Replaced by the hour (24-hour clock) as a decimal number [00,23].
%I	Replaced by the hour (12-hour clock) as a decimal number [00,12].
%j	Replaced by the day of the year as a decimal number [001,366].
%m	Replaced by the month as a decimal number [01,12].
%M	Replaced by the minute as a decimal number [00,59].
%n	Replaced by a newline character.
%p	Replaced by the locale equivalent of either A.M. or P.M. that the am_pm subcategory of LC_TIME defines.
%r	Indicates use of the format that the t_fmt_amp subcategory of LC_TIME defines.
%R	Replaced by the time in 24-hour notation (%H:%M).
%S	Replaced by the second as a decimal number.
%t	Replaced by a tab character.
%T	Replaced by the time (%H:%M:%S).

(2 of 3)

Formatting Directive	Description
%u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday.
%w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday.
%x	Indicates use of the format that the d_fmt subcategory of LC_TIME defines.
%X	Indicates use of the format that the t_fmt subcategory of LC_TIME defines.
%y	Replaced by year without century as a decimal number [00,99].
%Y	Replaced by year with century as a decimal number [00,99].
%%	Replaced by the % symbol.

(3 of 3)

If a formatting directive does not correspond to any of the preceding directives, the behavior of the formatting is undefined.

Modified Formatting Directives

You can modify some formatting directives with format modifiers, which follow the percent symbol (%), to indicate use of an alternative format that the LC_TIME locale-file category of the current locale might define. The **ifx_gl_format_datetime()** function supports the following format modifiers.

Format Modifier	Purpose	LC_TIME Subcategory
E	Use formats that include the era-based dates and times.	era, era_d_fmt, era_t_fmt, era_d_t_fmt
O	Use alternative locale-specific digits in dates and times.	alt_digits

The alternative format replaces one that an unmodified formatting directive normally uses. If the alternative format does not exist for the current locale, the behavior is the same as if unmodified formatting directives were used. The **ifx_gl_format_date()** function supports the following modified formatting directives with the **E** and **O** modifiers.

Formatting Directive	Description
%Ec	Indicates use of the format that the era_d_t_fmt subcategory of LC_TIME defines.
%EC	Replaced by the name of the base year (period) in the era subcategory of LC_TIME.
%Eg	Replaced by the abbreviated name of the base year (period) in the alternative representation that the era subcategory of LC_TIME defines.
%Ex	Indicates use of the format that the era_d_fmt subcategory of LC_TIME defines.
%EX	Indicates use of the format that the era_t_fmt subcategory of LC_TIME defines.
%Ey	Replaced by the offset from %EC (year only) in the alternative representation that the era subcategory of the LC_TIME defines.
%EY	Replaced by the full alternative year representation that the era subcategory of LC_TIME defines.
%Oe	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the day of the month. This formatting directive fills as needed with leading spaces.
%OH	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the hour (24-hour clock).
%OI	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the hour (12-hour clock).
%Om	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the month.
%OM	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the minutes.

(1 of 2)

Formatting Directive	Description
%OS	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the seconds.
%Ou	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the weekday as a number (Monday=1).
%Oy	Uses the alternative digits that the alt_digits subcategory of LC_TIME defines to format the value of %Ey .

(2 of 2)

The **ifx_gl_format_datetime()** function also supports the **i** modifier in the following formatting directives to support formats that are backward compatible with earlier Informix date and time formatting.

Formatting Directive	Description
%iF[n]	Replaced by the Informix DBTIME format %F . An optional precision specification, <i>n</i> , can follow the F . This <i>n</i> value must be between 1 and 5.
%iy	Replaced by the Informix DBDATE format Y2 . This format prints the two-digit year offset.
%iY	Replaced by the Informix DBDATE format Y4 . This format prints the full four-digit year.

Field Width and Precision

You can also specify an optional field width and precision in a formatting directive. This information precedes the type specifier in the formatting directive and has the following format:

`[- | 0][width][.precision]`

`[- | 0]` indicates field justification.

If the directive begins with a minus sign (-), the function left-justifies the field and pads it with spaces on the right.

If the directive begins with 0, the function right-justifies the field and pads it with zeros on the left.

Otherwise, the function right-justifies the field and pads it with spaces on the left.

width is a decimal value that specifies a minimum field width for the formatted *datetimestr* value.

precision has the format of a period (.) followed by a decimal value. The field *width* specifier can optionally be followed by a *precision* directive. For more information on the purpose of *precision*, see the following table.

The values of *width* and *precision* affect each element of the **%D**, **%R**, and **%T** directives. For example, **%6.4D** is interpreted as “%6.4m / %6.4d / %6.4y”.

The *precision* value has the following effects on the different formatting directives.

Formatting Directives	Effect of Precision
%C, %d, %e, %Ey, %F, %H, %iF, %iy, %iY, %I, %j, %m, %M, %S, %u, %w, %y, %Y	The value of <i>precision</i> specifies the minimum number of digits to appear. If a directive supplies fewer digits than the <i>precision</i> specifies, the function pads the value with leading zeros. The %d, %Ey, %H, %iF, %iy, %m, %M, %S, %u, %w, and %y directives have a default precision of 2. The %j directive has a default precision of 3. The %Y and %iY directives have a default precision of 4. The %F and %iF directives can be followed by an optional <i>precision</i> specification. For example, %F3 means a precision of 3. The value of this <i>precision</i> must be between 1 and 5. If it is not between 1 and 5, an error is returned. The %F directive has a default precision of 6. This directive overrides the <i>precision</i> value between the % symbol and the directive if one is given.
%a, %A, %b, %B, %EC, %Eg, %h, %p	The value of <i>precision</i> specifies the maximum number of characters to be used. If a value to be formatted has more characters than the <i>precision</i> specifies, the function truncates the result on the right.
%c, %Ec, %Ex, %EX, %EY, %n, %r, %t, %x, %X, %%	The values of <i>width</i> and <i>precision</i> do not affect these directives.
Directives modified with O (alternate digits)	The field <i>width</i> relates to display width rather than actual number of digits. The <i>precision</i> is still the minimum number of digits printed.

Locale Information

The LC_TIME category of the current locale affects the behavior of this function because it provides the locale-specific information for the formatting of the *datetime* value. For more information, see [“LC_TIME Locale-File Category” on page 3-6](#).

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_E2BIG	Formatting stopped due to lack of space in the buffer.
IFX_GL_EBADF	The formatting directive is invalid.
IFX_GL_EFRACRANGE	Fraction of second is out of bounds.
IFX_GL_ESECONDRANGE	Second is out of bounds.
IFX_GL_EMINUTERANGE	Minute is out of bounds.
IFX_GL_EHOURRANGE	Hour is out of bounds.
IFX_GL_EDAYRANGE	Day number is out of bounds.
IFX_GL_EWKDAYRANGE	Weekday number is out of bounds.
IFX_GL_EYDAYRANGE	Day-of-year number is out of bounds.
IFX_GLEMONTHRANGE	Month number is out of bounds.
IFX_GL_EYEARRANGE	Year number is out of bounds.
IFX_GL_EERAOFFRANGE	Era offset is out of bounds.
IFX_GL_BADFRAC	Fraction could not be scanned.
IFX_GL_BADSECOND	Second could not be scanned.
IFX_GL_BADMINUTE	Minute could not be scanned.
IFX_GL_BADHOUR	Hour could not be scanned.
IFX_GL_BADDAY	Month (as a number) could not be scanned.

Value	Description
IFX_GL_BADWKDAY	Weekday (as a number) could not be scanned.
IFX_GL_BADYDAY	Day of year (as a number) could not be scanned.
IFX_GL_BADMONTH	Month could not be scanned.
IFX_GL_BADYEAR	Year could not be scanned.
IFX_GL_BADERANAME	Era name was not found.
IFX_GL_BADERAOFFSET	Era offset could not be scanned.

(2 of 2)

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_datetime()**, **ifx_gl_format_date()**, and **ifx_gl_lc_errno()**.

ifx_gl_format_money()

The **ifx_gl_format_money()** function formats an internal money value to a money string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_format_money(monstr, monstrbytes, money, format)
    char *monstr;
    int monstrbytes;
    mi_money *money;
    char *format;
```

- monstr* is a pointer to the first character in the money string that the function formats from its internal *money* representation.
- monstrbytes* is the size of the *monstr* buffer for the money string. This size is the maximum size that the formatted string can reach.
- money* is a pointer to the variable that holds the internal representation that **ifx_gl_format_money()** formats to create the *monstr* money string.
- format* is a pointer to the format string that determines how to format the *monstr* money string. For more information, see [“Format String” on page 4-75](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_format_money()** function uses the format that the *format* string specifies to format the internal money representation in *money* as a money string. It stores the resulting money string in a buffer that *monstr* references.

The *money* argument is a pointer to a **decimal (dec_t)** value. ♦

The *money* argument is a pointer to an **mi_money** value. ♦

E/C

DB API

Format String

If *format* is NULL, the **ifx_gl_format_money()** function determines the format for the *monstr* money string that it creates from the environment, as follows:

- 1. If the **DBMONEY** environment variable is set, the function formats *monstr* according to **DBMONEY**.
- 2. Otherwise, the function uses the monetary formats in the LC_MONETARY category of the current GLS locale.

If *format* is not NULL, it must point to a string that follows the rules that this section describes. To format the *monstr* money string, **ifx_gl_format_money()** takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Formatting Action Taken
A valid formatting directive	The function performs the specified formatting of the data in the <i>money</i> argument. It replaces the formatting directive with a string representation of a money element for the <i>monstr</i> money string. Only one formatting directive is allowed in the <i>format</i> string.
Ordinary characters	The function copies the ordinary character unchanged to the <i>monstr</i> money string. Ordinary characters include the null terminator and white-space characters.

The formatting directive consists of the following sequence:

%[modifiers][flag][width[#left_precision][.right_precision]]type_specifier

Argument	Reference
<i>modifiers</i>	“Modified Formatting Directives” on page 4-77
<i>flag</i>	“Field Width and Precision” on page 4-78
<i>width</i>	“Field Width and Precision” on page 4-78



Argument	Reference
<i>left_precision</i>	“Field Width and Precision” on page 4-78
<i>right_precision</i>	“Field Width and Precision” on page 4-78
<i>type_specifier</i>	“Valid Type Specifiers” on page 4-76

(2 of 2)

Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter or letters within a formatting directive that identify a format for the **ifx_gl_format_money()** function to use when it creates a monetary element of a *monstr* string. These monetary formats are those that the LC_MONETARY locale-file category of the current locale might define. The **ifx_gl_format_money()** function supports the following formatting directives.

Formatting Directive	Description
%i	Replaced by the international monetary format (which uses the int_curr_symbol subcategory of the LC_MONETARY category in the current locale) for the <i>money</i> argument. For example, in the default locale, the international monetary format for 1,234.56 is as follows: USD 1,234.56
%n	Replaced by the national currency format (which uses the currency_symbol of the LC_MONETARY category in the current locale) for the <i>money</i> argument. For example, in the default locale, the national monetary format for 1,234.56 is as follows: \$1,234.56
%%	Replaced by the % symbol.

If a formatting directive does not correspond to any of the preceding directives, the behavior of the formatting is undefined.

Modified Formatting Directives

You can modify some formatting directives with format modifiers, which follow the percent symbol (%), to indicate use of an alternative format. The **ifx_gl_format_money()** function supports the following format modifiers.

Format Modifier	Description
<code>=f</code>	Use the single-byte character <i>f</i> as the numeric-fill character. The fill character must be representable in a single byte to work with precision and width counts. The default numeric-fill character is the blank (space) character. This flag does not affect field-width filling, which always uses the space character. Unless you specify <i>left_precision</i> , this flag is ignored.
<code>^</code>	Do not format the monetary amount with thousands separators. The default action is to insert the thousands separators if the <code>LC_MONETARY</code> category of the current locale defines them.
<code>+ or (</code>	Specify how to represent positive and negative monetary amounts. You can specify only <code>+</code> or <code>(</code> . If you specify the plus sign (<code>+</code>), the function uses the equivalent of <code>+</code> and <code>-</code> that the <code>LC_MONETARY</code> category of the current locale defines. For example, in the default locale, a plus sign means an the empty string for positive values and a minus sign (<code>-</code>) for negative values. If you do not specify either flag, the function uses the locale equivalent of the minus sign (<code>-</code>) for negative values and no sign for positive values.
<code>!</code>	Suppress the currency symbol from the formatted result.

Field Width and Precision

You can also specify an optional field width and precision in a formatting directive. This information precedes the type specifier in the formatting directive and has the following format:

```
[ - ][width][#left_precision][.right_precision]
```

-	indicates field justification. If the specification begins with a minus sign (-), the function left-justifies the field and pads it with spaces on the right. Otherwise, the function right-justifies the field and pads it with spaces on the left.
<i>width</i>	is a decimal value that specifies a minimum field width, in characters, in which the resulting string is right-justified. The default field width is 0.
<i>left_precision</i>	is a string that specifies the maximum number of digits expected to the left of the decimal separator. This option causes an amount to be formatted as if it has the number of digits that <i>n</i> specifies. If more than <i>n</i> digit positions are required, the function ignores this precision specification.
<i>right_precision</i>	is a decimal value that specifies the number of digits <i>after</i> the decimal separator. If the value of <i>right_precision</i> is 0, no decimal separator appears in the formatted string. If you do not include <i>right_precision</i> , the function uses a default, which the current locale specifies. The amount that is being formatted is rounded to the specified number of digits before formatting.

You can use this *left_precision* option to:

- keep the formatted output from multiple calls to this function aligned in the same columns.
- fill unused positions with a special character, such as the asterisk character in the “\$***123.45” string.

For *left_precision*, digit positions in excess of those actually required are filled with the numeric-fill character (see the description of the `=f` modifier in [“Modified Formatting Directives” on page 4-77](#)). If grouping has not been suppressed with the `^` modifier, and it is defined for the current locale, the function inserts thousands separators before the fill characters (if any) are added. Thousands separators are not applied to fill characters even if the fill character is a digit.

To ensure alignment, any characters that appear before or after the number in the formatted output, such as currency or sign symbols, are padded as necessary with space characters to make their positive and negative formats an equal length.

Locale Information

The LC_MONETARY category of the current locale affects the behavior of this function because it provides the locale-specific information for the formatting of the *money* value. For more information, see [“LC_MONETARY Locale-File Category” on page 3-5](#).

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_ENOSYS	The formatting directive is not supported.
IFX_GL_E2BIG	Formatting stopped due to insufficient space in the buffer.

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_money()**, **ifx_gl_format_number()**, and **ifx_gl_lc_errno()**.

ifx_gl_format_number()

The **ifx_gl_format_number()** function formats an internal decimal value to a number string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_format_number(numstr, numstrbytes, number, format)
    char *numstr;
    int numstrbytes;
    mi_decimal *number;
    char *format;
```

- numstr* is a pointer to the first character in the number string that the function formats from its internal *number* representation.
- numstrbytes* is the size of the *numstr* buffer for the number string. This size is the maximum size that the formatted string can reach.
- number* is a pointer to the variable that holds the internal representation that **ifx_gl_format_number()** formats to create the *numstr* number string.
- format* is a pointer to the format string that determines how to format the *numstr* number string. For more information, see [“Format String” on page 4-81](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_format_number()** function uses the format that the *format* string specifies to format the internal decimal representation in *number* as a number string. It stores the resulting number string in a buffer that *numstr* references.

The *number* argument is a pointer to a **decimal** (**dec_t**) value. ♦

The *number* argument is a pointer to an **mi_decimal** value. ♦

E/C

DB API

Format String

If *format* is NULL, the **ifx_gl_format_number()** function uses the numeric formats in the LC_NUMERIC category of the current GLS locale to format the *numstr* number string that it creates.

If *format* is not NULL, it must point to a string that follows the rules that this section describes. To format the *numstr* number string, the function **ifx_gl_format_number()** takes the following actions for each possible kind of character that it finds in *format*.

Contents of Format String	Formatting Action Taken
A valid formatting directive	The function performs the specified formatting of the data in the <i>number</i> argument. It replaces the formatting directive with a string representation of a numeric element for the <i>numstr</i> number string. Only one formatting directive is allowed in the <i>format</i> string.
Ordinary characters	The function copies the ordinary character unchanged to the <i>numstr</i> number string. Ordinary characters include the null terminator and white-space characters.

The formatting directive consists of the following sequence:

%[modifiers][flags][width[.precision]]type_specifier

Argument	Reference
<i>modifiers</i>	“Modified Formatting Directives” on page 4-83
<i>flags</i>	“Field Width and Precision” on page 4-84
<i>width</i>	“Field Width and Precision” on page 4-84
<i>precision</i>	“Field Width and Precision” on page 4-84
<i>type_specifier</i>	“Valid Type Specifiers” on page 4-82



Tip: In the preceding format sequence, the square brackets indicate that the enclosed portion of the format is optional.

Valid Type Specifiers

The type specifier is a letter or letters within a formatting directive that identify a format for the **ifx_gl_format_number()** function to use when it creates a numeric element of a *numstr* string. These numeric formats are those that the LC_NUMERIC locale-file category of the current locale might define. The **ifx_gl_format_number()** function supports the following formatting directives.

Formatting Directive	Description
%b	Replaced by the binary representation of <i>number</i> .
%d	Replaced by the decimal representation of <i>number</i> .
%e	Replaced by the style [-]d.ddde[+ -]dd of <i>number</i> (where <i>d</i> is a digit).
%E	Replaced by the style [-]d.dddE[+ -]dd of <i>number</i> (where <i>d</i> is a digit).
%f	Replaced by the style [-]ddd.ddd of <i>number</i> (where <i>d</i> is a digit). The default precision is 6.
%g	Replaced by the style of %f or %e . The %e directive is used only if the converted exponent is less than -4 or greater than or equal to the precision. The precision specifies the number of significant digits. Trailing zeros are removed from the fractional portion of the result.
%G	Is the same as %g , except for the replacement of %E for %e .
%i	Is the same as %d .
%o	Replaced by the octal representation of <i>number</i> .
%q	Replaced by the base-4 representation of <i>number</i> .
%u	Replaced by the unsigned decimal representation of <i>number</i> .
%x	Replaced by the hexadecimal representation of <i>number</i> , using the hexadecimal characters a through f.
%X	Is the same as %x , but use uppercase hexadecimal characters (A through F) instead of lowercase letters (a through f).

If a formatting directive does not correspond to any of the preceding directives, the behavior of the formatting is undefined.

Modified Formatting Directives

You can modify some formatting directives with format modifiers, which follow the percent symbol (%), to indicate use of an alternative format. The **ifx_gl_format_number()** function supports the following format modifiers.

Format Modifier	Description
'	Separates the significant digits of the converted number with the grouping character defined in the locale, according to the grouping format also defined in the locale.
+	Begins the result of a signed conversion with a positive or negative sign, which the locale defines.
-	Left-justifies the result of the conversion within the field.
space	Prefixes the result with a space character (unless the + modifier exists) if the beginning of a signed conversion is not a sign.
#	Converts the value to an alternate form. For a %o directive, the first digit of the result is forced to be 0. For a %x (or %X) directive, a nonzero result will have a leading "0x" (or "0X"). For a %e, %E, %f, %g, or %G directive, the result always has a decimal-separator character.
0	Pads the field width with leading zeros (unless the - modifier exists).

If the alternative format does not exist for the current locale, the function uses the unmodified formatting directive.

Field Width and Precision

You can also specify an optional field width and precision in a formatting directive. This information precedes the type specifier in the formatting directive and has the following format:

```
[ - | 0 ][width[.precision]]
```

[- | 0] indicates field justification.

If the specification begins with a minus sign (-), the function left-justifies the field and pads it with spaces on the right.

If the value begins with 0, the function right-justifies the field and pads it with zeros on the left.

Otherwise, the function right-justifies the field and pads it with spaces on the left.

width is a decimal value that specifies a minimum field width for the formatted string.

precision has the format of a period (.) followed by a decimal value. The field *width* specifier can optionally be followed by a *precision* directive.

For the **%d**, **%o**, **%u**, **%x**, and **%X** directives, the value of *precision* specifies the minimum number of digits to appear. If a directive supplies fewer digits than specified by the precision, it will be padded with leading zeros.

For the **%e**, **%E**, and **%f** directives, the value of *precision* specifies the number of digits to appear after the decimal separator.

For the **%g**, and **%G** directives, the value of *precision* specifies the maximum number of significant digits.

Locale Information

The LC_NUMERIC category of the current locale affects the behavior of this function because it provides the locale-specific information for the formatting of the *number* value. For more information, see [“LC_NUMERIC Locale-File Category” on page 3-4](#).

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_INVALIDFMT	The <i>format</i> string is invalid.
IFX_GL_PARAMERR	The type specifier in the formatting directive is invalid.
IFX_GL_E2BIG	Formatting stopped due to lack of space in the buffer.

Related Topics

See the descriptions of the following functions: **ifx_gl_convert_number()**, **ifx_gl_format_money()**, and **ifx_gl_lc_errno()**.

ifx_gl_getmb()

The **ifx_gl_getmb()** function obtains a single multibyte character from a user-specified location.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_fgetmb(mb, funcp, v, bytes_got)
    gl_mchar_t *mb;
    int (*funcp)(void *v);
    void *v;
    int *bytes_got;
```

- mb* is a pointer to the multibyte character whose bytes the *funcp* function reads from a specified location.
- funcp* is a pointer to a function that you define to specify the location from which to read the multibyte character.
- v* is a pointer that the **ifx_gl_getmb()** function passes to the *funcp* function each time that it is called.
- bytes_got* is a pointer to an integer that **ifx_gl_getmb()** sets to indicate the number of bytes that the *funcp* function has successfully read.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_getmb()** function calls a function that you define to obtain the bytes that form one multibyte character from a specified location. This multibyte character is then written to the *mb* buffer.

The *funcp* argument is a pointer to a function that you must define as follows:

```
int funcp(void *v)
```

The pointer *v* is passed to *funcp* each time that it is called. On success, *funcp* must return a value between 0 and 255, inclusive, which is the next byte of the multibyte character. On failure, *funcp* must return -1.

This function calls *funcp* until one of the following occurs:

- A complete multibyte character has been formed.
- The *funcp* function returns a byte that forms an illegal character when appended to the bytes already read.
- The *funcp* function fails.

All bytes obtained by calling *funcp* are guaranteed to be written to *mb*. This function assumes that *mb* is large enough to hold the result. You can determine what will be written to *mb* in either of the following ways:

- The function **ifx_gl_mb_loc_max()** calculates the maximum number of bytes in any multibyte character for the current locale.
- The macro **IFX_GL_MB_MAX** is the maximum number of bytes in any multibyte character in any locale. This value is always equal to or greater than the value that **ifx_gl_mb_loc_max()** returns.

Of these two options, the macro **IFX_GL_MB_MAX** is faster, and it can be used to initialize static buffers. The function **ifx_gl_mb_loc_max()** is slower but more precise.

The number of bytes that *funcp* successfully reads is returned in *bytes_got* (even when *funcp* fails).

Return Values

0 to 255	The value of the next byte for the multibyte character
-1	The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The bytes read from <i>funcp</i> do not form a valid multibyte character.
IFX_GL_EINVAL	The <i>funcp</i> function returned -1 in the middle of an otherwise valid multibyte character.
IFX_GL_EOF	The <i>funcp</i> function returned -1 on the first call.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mb_loc_max()**, and **ifx_gl_putmb()**.

ifx_gl_init()

The **ifx_gl_init()** function initializes the current processing locale for the current program.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_init(void)
```

Valid in Client Application	Valid in DataBlade UDR
Yes	Not needed

Usage

The **ifx_gl_init()** function initializes a global locale structure that identifies the current processing locale. Because this locale structure is global, the INFORMIX-GLS functions do not require the current locale as an argument. Any program that establishes its own connections must call the **ifx_gl_init()** function before it calls any other INFORMIX-GLS functions.

E/C

ESQL/C applications establish their own connections to a database server and therefore *must* call the **ifx_gl_init()** function before any INFORMIX-GLS functions to establish a current processing locale. ♦

DB API

DataBlade client applications establish their own connections to a database server and therefore *must* call the **ifx_gl_init()** function before any INFORMIX-GLS functions to establish a current processing locale.

IDS/UD

However, a DataBlade UDR executes in the context of an established connection and therefore has an established current processing locale. DataBlade UDRs are *not* required to call the **ifx_gl_init()** function. ♦

For both ESQL/C and DataBlade client applications, **ifx_gl_init()** initializes the current processing locale to the client locale.

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either specifiers for locale or code-set conversion (such as GLS environment variables) are incorrect, or the code-set name registry file could not be found.
IFX_GL_ENOMEM	Not enough memory is available to allocate a new locale object or a new code-set-conversion object.
IFX_GL_FILEERR	The locale or code-set conversion file could not be found, is not readable, or has the wrong format.
IFX_GL_INVALIDLOC	An attempt to create a locale with incompatible code sets occurred.
IFX_GL_ELOCTOOWIDE	The locale contains characters that are wider than the library allows.
IFX_GL_BADOBJVER	The locale object version is not compatible with the current library.
IFX_GL_BADFILEFORM	Bad format was found in the code-set registry file.

Related Topics

See [“Initializing the INFORMIX-GLS Library” on page 1-18.](#)

ifx_gl_ismalnum()

The **ifx_gl_ismalnum()** function determines whether a multibyte character contains an alphabetic or digit character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismalnum(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb*** is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit*** is the integer number of bytes to read from *mb* to try to form a complete multibyte character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismalnum()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismalnum()** function tests whether *mb* is in the alnum character class (either the alpha or digit class) according to the rules of the current locale. The alpha class contains all characters from the upper and lower classes. The digit class contains *only* the 10 ASCII digit characters: '0' to '9' (ASCII 0x030 to 0x039).

To determine whether a multibyte character is defined only in the alpha class, use the **ifx_gl_ismalpha()** function. To determine whether a multibyte character is defined only in the digit class, use the **ifx_gl_ismdigit()** function.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the alnum character class.

Return Values

- >0 The **mb* character is in the alpha or digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does *not* return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_isalpha()**, **ifx_gl_ismdigit()**, **ifx_gl_ismlower()**, **ifx_gl_ismupper()**, **ifx_gl_iswalnum()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismalpha()

The **ifx_gl_ismalpha()** function determines whether a multibyte character contains an alphabetic character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismalpha(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

mb is a pointer to the multibyte character whose character classification you want to determine.

mb_byte_limit is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismalpha()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismalpha()** function tests whether *mb* is in the alpha character class according to the rules of the current locale. The alpha class contains all characters from the upper and lower classes. In addition to the uppercase and lowercase Latin-based characters, this class includes any alphabetic characters that the locale might define, including:

- Asian ideographic characters; for example, Kanji characters.
- Asian phonetic characters; for example, single-byte and multibyte Katakana and Hiragana characters.
- non-ASCII digit characters (see the digit class).

- Latin-based alphabetic characters that do not have a case-equivalent character.
- user-defined characters. ♦
- vendor-defined characters.

Characters in the alpha class are also in the graph and print classes. No characters in the digit, blank, space, punct, or cntrl classes are in the alpha class.

To determine the case of an alphabetic multibyte character, you can use the **ifx_gl_ismupper()** and **ifx_gl_ismlower()** functions. Use the function **ifx_gl_ismalnum()** to test whether a multibyte character is an alphabetic character or a digit.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the alpha character class.

Return Values

- >0 The **mb* character is in the alpha character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_ismalnum()**, **ifx_gl_ismgraph()**, **ifx_gl_ismlower()**, **ifx_gl_ismprint()**, **ifx_gl_ismupper()**, **ifx_gl_iswalpha()**, **ifx_gl_lc_errno()**, **ifx_gl_tomlower()**, and **ifx_gl_tomupper()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismbblank()

The **ifx_gl_ismbblank()** function determines whether a multibyte character contains a horizontal-space character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismbblank(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismbblank()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismbblank()** function tests whether *mb* is in the blank character class according to the rules of the current locale. The blank character class includes the single-byte space (ASCII 0x020) and tab character (ASCII 0x009, ^I) plus any multibyte version of these characters that the locale defines. Characters in the blank class are also in the space class. No characters in the upper, lower, alpha, digit, xdigit, punct, or graph classes are in the blank class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the blank character class.

Return Values

- >0 The **mb* character is in the blank character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_ismspace()**, **ifx_gl_iswblank()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismcntrl()

The **ifx_gl_ismcntrl()** function determines whether a multibyte character contains a control character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismcntrl(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismcntrl()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismcntrl()** function tests whether *mb* is in the cntrl character class according to the rules of the current locale. The cntrl character class contains the single-byte control characters: 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, and DEL. These characters are the ASCII characters whose code points are in the range 0x00 to 0x1F. In addition, the cntrl class contains any other control characters that the locale might define.

No characters in the upper, lower, alpha, digit, xdigit, punct, graph, or print classes are in the cntrl class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the cntrl character class.

Return Values

- >0 The **mb* character is in the cntrl character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_iswcntrl()** and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismdigit()

The **ifx_gl_ismdigit()** function determines whether a multibyte character contains a decimal digit.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismdigit(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismdigit()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismdigit()** function tests whether *mb* is in the digit character class according to the rules of the current locale. The digit character class contains only the 10 ASCII digit characters (ASCII 0x030 to 0x039). Any multibyte versions or alternative representations of these digits that the locale might define (for example, Hindi or Kanji digits) are not in this class. Instead, they are in the alpha class.

Characters in this class are also in the *xdigit*, *graph*, and *print* classes. No characters in the upper, lower, alpha, blank, space, punct, or cntrl classes are in the digit class.

To determine whether a multibyte character contains a hexadecimal digit, you can use the **ifx_gl_ismxdigit()** function. Use the **ifx_gl_ismalnum()** function to test whether a multibyte character is an alphabetic character or a digit.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the digit character class.

Return Values

- >0 The **mb* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_ismalnum()**, **ifx_gl_ismalpha()**, **ifx_gl_ismgraph()**, **ifx_gl_ismprint()**, **ifx_gl_ismxdigit()**, **ifx_gl_iswdigit()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismgraph()

The **ifx_gl_ismgraph()** function determines whether a multibyte character contains a graphical (visible) character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismgraph(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

mb is a pointer to the multibyte character whose character classification you want to determine.

mb_byte_limit is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismgraph()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismgraph()** function tests whether *mb* is in the graph character class according to the rules of the current locale. The graph character class contains all characters that have a visual representation, including characters from the alpha, digit, punct, and xdigit classes. In addition, all placeholder characters used in round-trip code-set conversion are in this class.

Characters in the graph class are also in the print class. No characters in the blank, space, or cntrl classes are in the graph class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the graph character class.

Return Values

- >0 The **mb* character is in the graph character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_isalnum()**, **ifx_gl_isalpha()**, **ifx_gl_isdigit()**, **ifx_gl_islower()**, **ifx_gl_isprint()**, **ifx_gl_ispunct()**, **ifx_gl_isupper()**, **ifx_gl_isxdigit()**, **ifx_gl_iswgraph()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismlower()

The **ifx_gl_ismlower()** function determines whether a multibyte character contains a lowercase alphabetic character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismlower(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismlower()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismlower()** function tests whether *mb* is in the lower character class according to the rules of the current locale. The lower character class contains:

- the ASCII characters a through z.
- all other single-byte and multibyte lowercase characters for Latin-based languages; for example â, ç, é, and ü.

Characters in this class are also in the alpha, graph, and print classes. No characters in the upper, digit, blank, space, punct, or cntrl classes are in this class.

To determine the uppercase equivalent of an alphabetic multibyte character, you can use the **ifx_gl_toupper()** function. Use the **ifx_gl_ismalpha()** function to test whether a multibyte character is an alphabetic character (uppercase or lowercase).

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the lower character class.

Return Values

- >0 The **mb* character is in the lower character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_ismalnum()**, **ifx_gl_ismalpha()**, **ifx_gl_ismgraph()**, **ifx_gl_ismprint()**, **ifx_gl_ismupper()**, **ifx_gl_ismwlower()**, **ifx_gl_lc_errno()**, and **ifx_gl_toupper()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismprint()

The **ifx_gl_ismprint()** function determines whether a multibyte character contains a printable character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismprint(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismprint()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismprint()** function tests whether *mb* is in the print character class according to the rules of the current locale. The print character class contains all characters that have a visual representation or are in the space class but not in the cntrl class.

All characters from the alpha, digit, punct, xdigit, and graph classes are also in this class. No characters in the cntrl class are in this class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the print character class.

Return Values

- >0 The **mb* character is in the print character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_isalnum()**, **ifx_gl_isalpha()**, **ifx_gl_isdigit()**, **ifx_gl_ismgraph()**, **ifx_gl_ismlower()**, **ifx_gl_ismpunct()**, **ifx_gl_ismupper()**, **ifx_gl_ismxdigit()**, **ifx_gl_iswprint()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismpunct()

The **ifx_gl_ismpunct()** function determines whether a multibyte character contains a punctuation character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismpunct(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

mb is a pointer to the multibyte character whose character classification you want to determine.

mb_byte_limit is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismpunct()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismpunct()** function tests whether *mb* is in the punct character class according to the rules of the current locale. The punct character class contains the following characters:

- The single-byte ASCII punctuation characters: ! @ # \$ % ^ & * () - _ = + \ | ‘ ~ [] { } ; : ’ ” , . / < > ?
- Any non-ASCII punctuation characters that the locale might define

Graphic characters that are not really punctuation characters have traditionally been put in the punct class; instead they are in the graph class. Characters in this class are also in the graph and print classes. No characters in the upper, lower, alpha, digit, xdigit, blank, space, or cntrl classes are in this class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the punct character class.

Return Values

- >0 The **mb* character is in the punct character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_ismgraph()**, **ifx_gl_ismprint()**, **ifx_gl_iswpunct()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismspace()

The **ifx_gl_ismspace()** function determines whether a multibyte character contains a space (vertical or horizontal) character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismspace(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is `IFX_GL_NO_LIMIT`, the **ifx_gl_ismspace()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismspace()** function tests whether *mb* is in the space character class according to the rules of the current locale. The space character class contains:

- all characters from the blank class, including the horizontal space character (ASCII 0x020).
- the vertical-space character.

- the single-byte newline, vertical tab, form feed, and carriage return (ASCII 0x00A through 0x00D).
- any multibyte versions of newline, vertical tab, form feed, and carriage return.

No characters in the alpha, digit, xdigit, punct, or graph classes are in this class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the space character class.

Return Values

- >0 The **mb* character is in the space character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does *not* return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

ifx_gl_ismspace()

Related Topics

See the descriptions of the following functions: **ifx_gl_isblank()**, **ifx_gl_ismspace()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismupper()

The **ifx_gl_ismupper()** function determines whether a multibyte character contains an uppercase alphabetic character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismupper(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismupper()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismupper()** function tests whether *mb* is in the upper character class according to the rules of the current locale. The upper character class contains:

- the ASCII characters A through Z.
- all other single-byte and multibyte lowercase characters for Latin-based languages; for example: Â, Ç, É, and Û.

Characters in this class are also in the alpha, graph, and print classes. No characters in the lower, digit, blank, space, punct, or cntrl classes are in this class.

To determine the lowercase equivalent of an alphabetic multibyte character, you can use the **ifx_gl_tomlower()** function. Use the **ifx_gl_ismalpha()** function to test whether a multibyte character is an alphabetic character (uppercase or lowercase).

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the upper character class.

Return Values

- >0 The *mb* character is in the upper character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_ismalnum()**, **ifx_gl_ismalpha()**, **ifx_gl_ismgraph()**, **ifx_gl_ismlower()**, **ifx_gl_ismprint()**, **ifx_gl_ismupper()**, **ifx_gl_lc_errno()**, and **ifx_gl_tomupper()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_ismxdigit()

The **ifx_gl_ismxdigit()** function determines whether a multibyte character contains a hexadecimal digit.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_ismxdigit(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb* is a pointer to the multibyte character whose character classification you want to determine.
- mb_byte_limit* is the integer number of bytes to read from *mb* to try to form a complete character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismxdigit()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_ismxdigit()** function tests whether *mb* is in the xdigit character class according to the rules of the current locale. The xdigit character class contains:

- the 10 ASCII digit characters (ASCII 0x030 to 0x039).
- the characters A through F.
- the characters a through f.

Any multibyte versions or alternative representations of these hexadecimal digits that the locale might define (for example, Hindi or Kanji digits) are not in this class. Instead, they are in the alpha class.

Characters in this class are also in the graph and print classes. No characters in the blank, space, punct, or cntrl classes are in this class.

To determine whether a multibyte character contains a decimal digit, you can use the **ifx_gl_ismdigit()** function. Use the **ifx_gl_ismalnum()** function to test whether a multibyte character contains an alphabetic character *or* a digit.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the xdigit character class.

Return Values

- >0 The *mb* character is in the xdigit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_ismalnum()**, **ifx_gl_ismalpha()**, **ifx_gl_ismdigit()**, **ifx_gl_ismgraph()**, **ifx_gl_ismprint()**, **ifx_gl_iswxdigit()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswalnum()

The **ifx_gl_iswalnum()** function determines whether a wide character contains an alphabetic or digit character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswalnum(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswalnum()** function tests whether *wc* is in the alnum character class (either the alpha or digit class) according to the rules of the current locale. The alpha class contains all characters from the upper and lower classes. The digit class contains only the 10 ASCII digit characters: '0' to '9' (ASCII 0x030 to 0x039).

To determine whether a wide character is defined only in the alpha class, use the **ifx_gl_iswalpha()** function. To determine whether a wide character is defined only in the digit class, use the **ifx_gl_iswdigit()** function.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the alnum character class.

Return Values

- >0 The `wc` character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does *not* return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <code>*wc</code> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismalnum()**, **ifx_gl_iswalpha()**, **ifx_gl_iswdigit()**, **ifx_gl_iswlower()**, **ifx_gl_iswupper()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswalpha()

The **ifx_gl_iswalpha()** function determines whether a wide character contains an alphabetic character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswalpha(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswalpha()** function tests whether *wc* is in the alpha character class according to the rules of the current locale. The alpha class contains all characters from the upper and lower classes. In addition to the uppercase and lowercase Latin-based characters, this class includes any alphabetic characters that the locale might define, including:

- Asian ideographic characters; for example, Kanji characters.
- Asian phonetic characters; for example, single-byte and multibyte Katakana and Hiragana characters.
- non-ASCII digit characters (see the digit class).
- Latin-based alphabetic characters that do not have a case-equivalent character.
- user-defined characters. ♦
- vendor-defined characters.

Characters in the alpha class are also in the graph and print classes. No characters in the digit, blank, space, punct, or cntrl classes are in this class.

To determine the case of a wide alphabetic character, you can use the **ifx_gl_iswupper()** and **ifx_gl_iswlower()** functions. Use the **ifx_gl_iswalnum()** function to test whether a wide character is an alphabetic character or a digit.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the alpha character class.

Return Values

- >0 The *wc* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <i>*wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismalpha()**, **ifx_gl_iswalnum()**, **ifx_gl_iswgraph()**, **ifx_gl_iswlower()**, **ifx_gl_iswprint()**, **ifx_gl_iswupper()**, **ifx_gl_lc_errno()**, **ifx_gl_towlower()**, and **ifx_gl_towupper()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswblank()

The **ifx_gl_iswblank()** function determines whether a wide character contains a horizontal-space character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswblank(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswblank()** function tests whether *wc* is in the blank character class according to the rules of the current locale. The blank character class includes the single-byte space (ASCII 0x020) and tab character (ASCII 0x009, ^I) plus any multibyte version of these characters that the locale defines. Characters in the blank class are also in the space class. No characters in the upper, lower, alpha, digit, xdigit, punct, or graph classes are in the blank class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the blank character class.

Return Values

- >0 The `wc` character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <code>*wc</code> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismbblank()**, **ifx_gl_iswspace()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswcntrl()

The **ifx_gl_iswcntrl()** function determines whether a wide character contains a control character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswcntrl(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswcntrl()** function tests whether *wc* is in the *cntrl* character class according to the rules of the current locale. The *cntrl* character class contains the single-byte control characters: 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, and DEL. These characters are the ASCII characters whose code points are in the range 0x000 to 0x1F. In addition, the *cntrl* class contains any other control characters that the locale might define.

No characters in the upper, lower, alpha, digit, xdigit, punct, graph, or print classes are in the *cntrl* class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the *cntrl* character class.

Return Values

- >0 The `wc` character is in the `cntrl` character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <code>*wc</code> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismcntrl()** and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswdigit()

The **ifx_gl_iswdigit()** function determines whether a wide character contains a decimal digit.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswdigit(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswdigit()** function tests whether *wc* is in the digit character class according to the rules of the current locale. The digit character class contains only the 10 ASCII digit characters (ASCII 0x030 to 0x039). Any multibyte versions or alternative representations of these digits that the locale might define (for example, Hindi or Kanji digits) are not in this class. Instead, they are in the alpha class.

Characters in this class are also in the *xdigit*, *graph*, and *print* classes. No characters in the *upper*, *lower*, *alpha*, *blank*, *space*, *punct*, or *cntrl* classes are in this class.

To determine whether a wide character contains a hexadecimal digit, you can use the **ifx_gl_iswxdigit()** function. Use the **ifx_gl_iswalnum()** function to test whether a wide character is an alphabetic character or a digit.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the digit character class.

Return Values

- >0 The *wc* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The * <i>wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismdigit()**, **ifx_gl_iswalnum()**, **ifx_gl_iswalpha()**, **ifx_gl_iswgraph()**, **ifx_gl_iswprint()**, **ifx_gl_iswxdigit()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswgraph()

The **ifx_gl_iswgraph()** function determines whether a wide character contains a graphical (visible) character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswgraph(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswgraph()** function tests whether *wc* is in the graph character class according to the rules of the current locale. The graph character class contains all characters that have a visual representation, including characters from the alpha, digit, punct, and xdigit classes. In addition, all placeholder characters used in round-trip code-set conversion are in this class.

Characters in the graph class are also in the print class. No characters in the blank, space, or cntrl classes are in the graph class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the graph character class.

Return Values

- >0 The `wc` character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does *not* return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <code>*wc</code> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismgraph()**, **ifx_gl_iswalnum()**, **ifx_gl_iswalpha()**, **ifx_gl_iswdigit()**, **ifx_gl_iswlower()**, **ifx_gl_iswprint()**, **ifx_gl_iswpunct()**, **ifx_gl_iswupper()**, **ifx_gl_iswxdigit()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswlower()

The **ifx_gl_iswlower()** function determines whether a wide character contains a lowercase alphabetic character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswlower(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswlower()** function tests whether *wc* is in the lower character class according to the rules of the current locale. The lower character class contains:

- the ASCII characters a through z.
- all other single-byte and multibyte lowercase characters for Latin-based languages; for example: â, ç, é, and ü.

Characters in this class are also in the alpha, graph, and print classes. No characters in the upper, digit, blank, space, punct, or cntrl classes are in this class.

To obtain the uppercase equivalent of an alphabetic wide character, you can use the **ifx_gl_toupper()** function. Use the **ifx_gl_iswalpha()** function to test whether a wide character is an alphabetic character (uppercase or lowercase).

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the lower character class.

Return Values

- >0 The *wc* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The * <i>wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismlower()**, **ifx_gl_iswalnum()**, **ifx_gl_iswalpha()**, **ifx_gl_iswgraph()**, **ifx_gl_iswprint()**, **ifx_gl_iswupper()**, **ifx_gl_lc_errno()**, and **ifx_gl_towupper()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswprint()

The **ifx_gl_iswprint()** function determines whether a wide character contains a printable character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswprint(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswprint()** function tests whether *wc* is in the print character class according to the rules of the current locale. The print character class contains all characters that have a visual representation or are in the space class but not in the cntrl class.

All characters from the alpha, digit, punct, xdigit, and graph classes are also in this class. No characters in the cntrl class are in this class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the print character class.

Return Values

- >0 The `wc` character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <code>*wc</code> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismprint()**, **ifx_gl_iswalnum()**, **ifx_gl_iswalpha()**, **ifx_gl_iswdigit()**, **ifx_gl_iswgraph()**, **ifx_gl_iswlower()**, **ifx_gl_iswpunct()**, **ifx_gl_iswupper()**, **ifx_gl_iswxdigit()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswpunct()

The **ifx_gl_iswpunct()** function determines whether a wide character contains a punctuation character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswpunct(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswpunct()** function tests whether *wc* is in the punct character class according to the rules of the current locale. The punct character class contains the following characters:

- The single-byte ASCII punctuation characters: ! @ # \$ % ^ & * () - _ = + \ | ' ~ [] { } ; : ' " , . / < > ?
- Any non-ASCII punctuation characters that the locale might define

Graphic characters that are not true punctuation characters have traditionally been put in the punct class; instead they are in the graph class. Characters in this class are also in the graph and print classes. No characters in the upper, lower, alpha, digit, xdigit, blank, space, or cntrl classes are in this class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the punct character class.

Return Values

- >0 The *wc* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The * <i>wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismpunct()**, **ifx_gl_iswgraph()**, **ifx_gl_iswprint()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswspace()

The **ifx_gl_iswspace()** function determines whether a wide character contains a space (vertical or horizontal) character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswspace(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswspace()** function tests whether **wc** is in the space character class according to the rules of the current locale. The space character class contains:

- all characters from the blank class, including the horizontal space character (ASCII 0x020).
- the vertical-space character.
- the single-byte newline, vertical tab, form feed, and carriage return (ASCII 0x0A through 0x0D).
- any multibyte versions of newline, vertical tab, form feed, and carriage return.

No characters in the alpha, digit, xdigit, punct, or graph classes are in this class.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the space character class.

Return Values

- >0 The *wc* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The * <i>wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismspace()**, **ifx_gl_iswblank()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswupper()

The **ifx_gl_iswupper()** function determines whether a wide character contains an uppercase alphabetic character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswupper(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswupper()** function tests whether *wc* is in the upper character class according to the rules of the current locale. The upper character class contains:

- the ASCII characters A through Z.
- all other single-byte and multibyte uppercase characters for Latin-based languages; for example: Â, Ç, É, and Û.

Characters in this class are also in the alpha, graph, and print classes. No characters in the lower, digit, blank, space, punct, or cntrl classes are in this class.

To determine the lowercase equivalent of an alphabetic wide character, you can use the **ifx_gl_tomlower()** function. Use the **ifx_gl_ismalpha()** function to test whether a wide character is an alphabetic character (uppercase or lowercase).

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the upper character class.

Return Values

- >0 The *wc* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The * <i>wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismupper()**, **ifx_gl_iswalnum()**, **ifx_gl_iswalpha()**, **ifx_gl_iswgraph()**, **ifx_gl_iswlower()**, **ifx_gl_iswprint()**, **ifx_gl_lc_errno()**, and **ifx_gl_towupper()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_iswxdigit()

The **ifx_gl_iswxdigit()** function determines whether a wide character contains a hexadecimal digit.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_iswxdigit(wc)
    gl_wchar_t *wc;
```

wc is a pointer to the wide character whose character classification you want to determine.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_iswxdigit()** function tests whether *wc* is in the *xdigit* character class according to the rules of the current locale. The *xdigit* character class contains:

- the 10 ASCII digit characters (ASCII 0x030 to 0x039).
- the characters A through F.
- the characters a through f.

Any multibyte versions or alternative representations of these hexadecimal digits that the locale might define (for example, Hindi or Kanji digits) are not in this class. Instead, they are in the *alpha* class.

Characters in this class are also in the *graph* and *print* classes. No characters in the *blank*, *space*, *punct*, or *cntrl* classes are in this class.

To determine whether a wide character contains a decimal digit, you can use the **ifx_gl_iswdigit()** function. Use the **ifx_gl_iswalnum()** function to test whether a wide character contains an alphabetic character *or* a digit.

Locale Information

The LC_CTYPE category of the current locale affects the behavior of this function because it defines the xdigit character class.

Return Values

- >0 The *wc* character is in the digit character class.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does *not* return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The *wc value is not a valid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_ismxdigit()**, **ifx_gl_iswalnum()**, **ifx_gl_iswalpha()**, **ifx_gl_iswdigit()**, **ifx_gl_iswgraph()**, **ifx_gl_iswprint()**, and **ifx_gl_lc_errno()**.

For a description of character classification, see [“Character Classification” on page 2-11](#).

ifx_gl_lc_errno()

The **ifx_gl_lc_errno()** function returns the value of the error number that some other INFORMIX-GLS function has set.

Syntax

```
#include <gls.h>
...
int ifx_gl_lc_errno(void);
```

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The INFORMIX-GLS functions set the *error number*, of type **int**, to indicate additional information about an error that occurred. INFORMIX-GLS functions set this value only when an error occurs (unless documented otherwise). Because any INFORMIX-GLS function can set the error number, you must use **ifx_gl_lc_errno()** to inspect the error number immediately after you call the function in which the error occurred.

Tip: The *Errors* section for each function documents the values that **ifx_gl_lc_errno()** returns after that particular function.

Certain code-set conversion functions are not dependent on a locale and do not set an error number that you can retrieve with **ifx_gl_lc_errno()**. Instead, these functions return a negative error code.



Return Values

Integer error number	The error number that an INFORMIX-GLS function has set. These error numbers are defined in the gls.h file and are described in the Errors section for each function.
-1	The function was not successful. The current processing locale has not been correctly initialized.

Errors

None

Related Topics

See [“Handling INFORMIX-GLS Exceptions” on page 1-19](#).

ifx_gl_mb_loc_max()

The **ifx_gl_mb_loc_max()** function returns the maximum number of bytes in the characters of the current locale.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mb_loc_max(void);;
```

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mb_loc_max()** function is useful when you need to allocate memory for a multibyte-character string. It returns the maximum number of bytes that any multibyte character can occupy. A multibyte code set can have characters of one, two, three, or four bytes.

Return Values

- >=0 The maximum number of bytes in the current locale.
- 1 The function was not successful.

Errors

None

Related Topics

See [“Managing Memory for Strings and Characters” on page 2-41](#).

ifx_gl_mblen()

The **ifx_gl_mblen()** function returns the number of bytes in a multibyte character.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mblen(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb** is a pointer to the multibyte character whose length you want to determine.
- mb_byte_limit** is the integer number of bytes to read from *mb* to try to form a complete multibyte character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, the **ifx_gl_ismalnum()** function reads as many bytes as necessary from *mb* to form a complete character. For more information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mblen()** function determines the number of bytes in the multibyte character, *mb*. This function is often used to advance a pointer through a null-terminated multibyte string. However, it is slightly faster to advance a pointer through a null-terminated string with **ifx_gl_mbsnext()**, as the following code shows:

```
for ( mb = mbs; *mb != '\0'; )
{
    if ( (mb = ifx_gl_mbsnext(mb, IFX_GL_NO_LIMIT)) == NULL )
        /* handle error */
}
```

Return Values

- >=0 The number of bytes in the *mb* multibyte character.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsnext()**, and **ifx_gl_mbsprev()**.

For more information, see [“String Traversal” on page 2-30](#).

ifx_gl_mbscat()

The **ifx_gl_mbscat()** function concatenates two multibyte-character strings.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbscat(mbs1, mbs1_byte_length, mbs2, mbs2_byte_length)
    gl_mchar_t *mbs1;
    int mbs1_byte_length;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
```

- mbs1*

is a pointer to the multibyte-character string to which the function concatenates *mbs2*.
- mbs1_byte_length*

is the integer number of bytes in the *mbs1* string. If *mbs1_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs1* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).
- mbs2*

is a pointer to the multibyte-character string to concatenate onto *mbs1*.
- mbs2_byte_length*

is the integer number of bytes in the *mbs2* string. If *mbs2_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs2* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbscat()** function appends a copy of the *mbs2* multibyte-character string to the end of the *mbs1* multibyte-character string. If *mbs1* and *mbs2* overlap, the results of this function are undefined.

If *mbs1_byte_length* is equal to `IFX_GL_NULL`, the function null-terminates the concatenated string. Any other value of *mbs1_byte_length* means that the function does not null-terminate the resulting concatenated string.

Return Values

- >=0 The number of characters in the concatenated string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>mbs1_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0, or <i>mbs2_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>mbs1_byte_length</i> is equal to <code>IFX_GL_NULL</code> , and <i>mbs2_byte_length</i> is greater than or equal to 0; or <i>mbs1_byte_length</i> is greater than or equal to 0, and <i>mbs2_byte_length</i> is equal to <code>IFX_GL_NULL</code> .

(1 of 2)

Value	Description
IFX_GL_EILSEQ	Either <i>mbs1</i> or <i>mbs2</i> contains an invalid multibyte character.
IFX_GL_EINVAL	Either the function cannot determine whether the last character of <i>mbs1</i> is a valid multibyte character because it would need to read more than <i>mbs1_byte_length</i> bytes from <i>mbs1</i> , or the function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

(2 of 2)

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsncat()**, **ifx_gl_wcscat()**, and **ifx_gl_wcsncat()**.

For more information, see [“Concatenation” on page 2-32](#).

ifx_gl_mbschr()

The **ifx_gl_mbschr()** function searches for the first occurrence of a character in a multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
gl_mchar_t *ifx_gl_mbschr(mbs, mbs_byte_length, mb, mb_byte_limit)
    gl_mchar_t *mbs;
    int mbs_byte_length;
    gl_mchar_t *mb;
    int mb_byte_limit;
```

<i>mbs</i>	is a pointer to the multibyte-character string in which the function searches for <i>mb</i> .
<i>mbs_byte_length</i>	is the integer number of bytes in the <i>mbs</i> string. If <i>mbs_byte_length</i> is the value <code>IFX_GL_NULL</code> , the function assumes that <i>mbs</i> is a null-terminated string. For more general information about string lengths, see “Character-String Termination” on page 2-38 .
<i>mb</i>	is a pointer to the multibyte character to search for in <i>mbs</i> .
<i>mb_byte_limit</i>	is the number of bytes to read from the <i>mb</i> character to try to form a complete multibyte character. If <i>mb_byte_limit</i> is <code>IFX_GL_NO_LIMIT</code> , this function reads as many bytes as necessary from <i>mb</i> to form a complete character. For more information about <i>mb_byte_limit</i> , see “Multibyte-Character Termination” on page 2-39 .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbschr()** function locates the first occurrence of *mb* in the multibyte string *mbs*.

Return Values

gl_mchar_t *	A pointer to the first occurrence of <i>mb</i> in <i>mbs</i> .
NULL	Either <i>mb</i> was not found in <i>mbs</i> (the ifx_gl_lc_errno() error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is an invalid multibyte character, or <i>*mbs</i> contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> , or the function cannot determine whether the last character of <i>mbs</i> is a valid multibyte character because it would need to read more than <i>mbs_byte_length</i> bytes from <i>mbs</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .
IFX_GL_PARAMERR	The <i>mbs_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsrchr()**, **ifx_gl_wcschr()**, and **ifx_gl_wcsrchr()**.

For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_mbscoll()

The **ifx_gl_mbscoll()** function uses localized collation order to compare two multibyte-character strings.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbscoll(mbs1, mbs1_byte_length, mbs2, mbs2_byte_length)
    gl_mchar_t *mbs1;
    int mbs1_byte_length;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
```

- mbs1*

is a pointer to the multibyte-character string to compare with *mbs2*.
- mbs1_byte_length*

is the integer number of bytes in the *mbs1* string. If *mbs1_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs1* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).
- mbs2*

is a pointer to the string of multibyte characters to compare with *mbs1*.
- mbs2_byte_length*

is the integer number of bytes in the *mbs2* string. If *mbs2_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs2* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbscoll()** function uses localized collation order to compare the multibyte-character strings *mbs1* and *mbs2*.

Locale Information

The LC_COLLATE category of the current locale affects the behavior of this function because it defines the localized collation order. For more information, see [“Character/String Comparison and Sorting” on page 2-35](#).

Return Values

- <0 The *mbs1* string is less than the *mbs2* string ($mbs1 < mbs2$); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.
- =0 The *mbs1* string is equal to the *mbs2* string ($mbs1 = mbs2$).
- >0 The *mbs1* string is greater than the *mbs2* string ($mbs1 > mbs2$).

Errors

This function does *not* return a unique value to indicate an error. If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>mbs1_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0; or <i>mbs2_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>mbs1_byte_length</i> is equal to IFX_GL_NULL, and <i>mbs2_byte_length</i> is greater than or equal to 0; or <i>mbs1_byte_length</i> is greater than or equal to 0, and <i>mbs2_byte_length</i> is equal to IFX_GL_NULL.
IFX_GL_EILSEQ	Either <i>mbs1</i> or <i>mbs2</i> contains an invalid multibyte character.

(1 of 2)

Value	Description
IFX_GL_EINVAL	Either the function cannot determine whether the last character of <i>mbs1</i> is a valid multibyte character because it would need to read more than <i>mbs1_byte_length</i> bytes from <i>mbs1</i> , or the function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .
IFX_GL_ENOMEM	Not enough memory is available to complete the operation.

(2 of 2)

Related Topics

See the descriptions of the following functions: **`ifx_gl_lc_errno()`** and **`ifx_gl_wcscoll()`**.

For more information, see [“Character/String Comparison and Sorting” on page 2-35](#).

ifx_gl_mbscopy()

The **ifx_gl_mbscopy()** function copies a multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbscopy(mbs1, mbs2, mbs2_byte_length)
    gl_mchar_t *mbs1;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
```

- mbs1* is a pointer to the location where the function copies *mbs2*.
- mbs2* is a pointer to the multibyte-character string to copy to *mbs1*.
- mbs2_byte_length* is the integer number of bytes in the *mbs2* string. If *mbs2_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs2* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbscopy()** function copies the multibyte string *mbs2* to the location that *mbs1* references. If *mbs1* and *mbs2* overlap, the results of this function are undefined.

If *mbs2_byte_length* is equal to IFX_GL_NULL, the function null-terminates the concatenated string. Any other value of *mbs2_byte_length* means that the function does not null-terminate the resulting concatenated string.

Return Values

- >=0 The number of bytes in the copied string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	The <i>mbs2_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_EILSEQ	The <i>*mbs2</i> value contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsncpy()**, **ifx_gl_wcscpy()**, and **ifx_gl_wcsncpy()**.

For more information, see [“String Copying”](#) on page 2-33.

ifx_gl_mbscspn()

The **ifx_gl_mbscspn()** function determines the length of a complementary multibyte substring for a multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbscspn(mbs1, mbs1_byte_length, mbs2, mbs2_byte_length)
    gl_mchar_t *mbs1;
    int mbs1_byte_length;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
```

<i>mbs1</i>	is a pointer to the multibyte-character string to check for the complementary string of characters in <i>mbs2</i> .
<i>mbs1_byte_length</i>	is the integer number of bytes in the <i>mbs1</i> string. If <i>mbs1_byte_length</i> is the value IFX_GL_NULL, the function assumes that <i>mbs1</i> is a null-terminated string. For more general information about string lengths, see “Character-String Termination” on page 2-38 .
<i>mbs2</i>	is a pointer to the string of multibyte characters to search for in <i>mbs1</i> .
<i>mbs2_byte_length</i>	is the integer number of bytes in the <i>mbs2</i> string. If <i>mbs2_byte_length</i> is the value IFX_GL_NULL, the function assumes that <i>mbs2</i> is a null-terminated string. For more general information about string lengths, see “Character-String Termination” on page 2-38 .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbscspn()** function returns the size of the initial substring of *mbs1* that does not contain any characters that match characters in *mbs2*. The initial substring begins at the first character of *mbs1*. Therefore, this size is the number of characters in *mbs1* before the first character that the *mbs1* and *mbs2* strings have in common. The string of unmatched characters is called the *complementary string*.

For example, suppose you have the following two multibyte-characters strings, **mbs1** and **mbs2**:

```
mbs1 = "A1A2B1B2B3C1C2D1D2D3A1A2E1F1F2A1A2G1D1D2D3";  
mbs2 = "F1F2G1D1D2D3";
```

With these multibyte strings, the following call to the **ifx_gl_mbscspn()** function returns 3:

```
ifx_gl_mbscspn(mbs1, bytelen1, mbs2, bytelen2)
```

The first three characters of **mbs1** are not in **mbs2**. The fourth character in **mbs1** is D¹D²D³, which is in **mbs2** also.

Return Values

- >=0 The number of characters in the initial substring of *mbs1* that consists entirely of multibyte characters not in the string *mbs2*.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>mbs1_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0; or <i>mbs2_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>mbs1_byte_length</i> is equal to IFX_GL_NULL, and <i>mbs2_byte_length</i> is greater than or equal to 0; or <i>mbs1_byte_length</i> is greater than or equal to 0, and <i>mbs2_byte_length</i> is equal to IFX_GL_NULL.
IFX_GL_EILSEQ	Either <i>mbs1</i> or <i>mbs2</i> contains an invalid multibyte character.
IFX_GL_EINVAL	Either the function cannot determine whether the last character of <i>mbs1</i> is a valid multibyte character because it would need to read more than <i>mbs1_byte_length</i> bytes from <i>mbs1</i> , or the function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsspn()**, **ifx_gl_wcssp()**, and **ifx_gl_wcsspn()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_mbslen()

The **ifx_gl_mbslen()** function determines the number of characters in a multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbslen(mbs, mbs_byte_length)
    gl_mchar_t *mbs;
    int mbs_byte_length;
```

- mbs* is a pointer to the multibyte-character string whose length the function determines.
- mbs_byte_length* is the integer number of bytes in the *mbs* string. If *mbs_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbslen()** function returns the number of characters (not bytes) in the string *mbs*, not including any terminating null character. The length that **ifx_gl_mbslen()** returns includes any trailing white space. The trailing-space characters are all characters that the blank character class of the current locale defines. Use the **ifx_gl_mbntslen()** function if you want the length without trailing white space.

For example, the following call to **ifx_gl_mbslen()** return a value of 12 (where A¹A², B¹B²B³, and E¹ are multibyte characters of 2, 3, and 1 bytes, respectively, and each s represents a single-byte horizontal white-space character):

```
ifx_gl_mbslen("A1A2B1B2B3scdsE1sssss")
```

Use the **ifx_gl_mbsntslen()** or **ifx_gl_mbsntsbytes()** function if you want the length without trailing white space.

Return Values

- >=0 The number of characters in the *mbs* string, not including any null terminator but including any trailing white space.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	The <i>mbs_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_EILSEQ	The <i>*mbs</i> value contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether the last character of <i>mbs</i> is a valid multibyte character because it would need to read more than <i>mbs_byte_length</i> bytes from <i>mbs</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mblen()**, **ifx_gl_mbsnext()**, **ifx_gl_mbsntsbytes()**, **ifx_gl_mbsntslen()**, and **ifx_gl_wcslen()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_mbsmbs()

The **ifx_gl_mbsmbs()** function searches for a specified substring within a multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
gl_mchar_t *ifx_gl_mbsmbs(mbs1, mbs1_byte_length, mbs2, mbs2_byte_length)
    gl_mchar_t *mbs1;
    int mbs1_byte_length;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
```

- mbs1*

is a pointer to the multibyte-character string to search for the substring *mbs2*.
- mbs1_byte_length*

is the integer number of bytes in the *mbs1* string. If *mbs1_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs1* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).
- mbs2*

is a pointer to the string of multibyte characters to search for in *mbs1*.
- mbs2_byte_length*

is the integer number of bytes in the *mbs2* string. If *mbs2_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs2* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsmbs()** function searches for the first occurrence of the multibyte string *mbs2* in the multibyte string *mbs1*.

Return Values

<code>gl_mchar_t *</code>	A pointer to the first byte of the first occurrence of <i>mbs2</i> in <i>mbs1</i> .
<code>NULL</code>	Either <i>mbs2</i> was not found in <i>mbs1</i> (the ifx_gl_lc_errno() error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets **ifx_gl_lc_errno()** to one of the following values.

Value	Description
<code>IFX_GL_PARAMERR</code>	Either <i>mbs1_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0, or <i>mbs2_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0.
<code>IFX_GL_TERMMISMAT</code>	Either <i>mbs1_byte_length</i> is equal to <code>IFX_GL_NULL</code> , and <i>mbs2_byte_length</i> is greater than or equal to 0; or <i>mbs1_byte_length</i> is greater than or equal to 0, and <i>mbs2_byte_length</i> is equal to <code>IFX_GL_NULL</code> .
<code>IFX_GL_EILSEQ</code>	Either <i>mbs1</i> or <i>mbs2</i> contains an invalid multibyte character.
<code>IFX_GL_EINVAL</code>	Either the function cannot determine whether the last character of <i>mbs1</i> is a valid multibyte character because it would need to read more than <i>mbs1_byte_length</i> bytes from <i>mbs1</i> , or the function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43.

ifx_gl_mbsmbs()

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbschr()**, **ifx_gl_mbsrchr()**, **ifx_gl_wcschr()**, **ifx_gl_wcsrchr()**, and **ifx_gl_wcswcs()**.

For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_mbsncat()

The **ifx_gl_mbsncat()** function concatenates a specified number of multibyte characters in one multibyte string to another.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbsncat(mbs1, mbs1_byte_length, mbs2, mbs2_byte_length, char_limit)
    gl_mchar_t *mbs1;
    int mbs1_byte_length;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
    int char_limit;
```

- mbs1*

is a pointer to the multibyte-character string to which the function concatenates *mbs2*.
- mbs1_byte_length*

is the integer number of bytes in the *mbs1* string. If *mbs1_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs1* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).
- mbs2*

is a pointer to the multibyte-character string to concatenate to *mbs1*.
- mbs2_byte_length*

is the integer number of bytes in the *mbs2* string. If *mbs2_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs2* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).
- char_limit*

is the maximum number of multibyte characters to read from *mbs2*.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsncat()** function appends *mbs2* to the end of *mbs1*. The function reads no more than *char_limit* characters from *mbs2* and writes them to *mbs1*. If *mbs1* and *mbs2* overlap, the results of this function are undefined.

If *mbs1_byte_length* is equal to IFX_GL_NULL, the function null-terminates the concatenated string. Any other value of *mbs1_byte_length* means that the function does not null-terminate the resulting concatenated string.

Return Values

- >=0 The number of bytes in the concatenated string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>mbs1_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0, or <i>mbs2_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>mbs1_byte_length</i> is equal to IFX_GL_NULL, and <i>mbs2_byte_length</i> is greater than or equal to 0; or <i>mbs1_byte_length</i> is greater than or equal to 0, and <i>mbs2_byte_length</i> is equal to IFX_GL_NULL.

(1 of 2)

Value	Description
IFX_GL_EILSEQ	Either <i>mbs1</i> or <i>mbs2</i> contains an invalid multibyte character.
IFX_GL_EINVAL	Either the function cannot determine whether the last character of <i>mbs1</i> is a valid multibyte character because it would need to read more than <i>mbs1_byte_length</i> bytes from <i>mbs1</i> , or the function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

(2 of 2)

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscat()**, **ifx_gl_wcscat()**, and **ifx_gl_wcsncat()**.

For more information, see [“Concatenation” on page 2-32](#).

ifx_gl_mbsncpy()

The **ifx_gl_mbsncpy()** function copies a specified number of multibyte characters from a multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbsncpy(mbs1, mbs2, mbs2_byte_length, char_limit)
    gl_mchar_t *mbs1;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
    int char_limit;
```

- mbs1*

is a pointer to the location where the function copies *mbs2*.
- mbs2*

is a pointer to the multibyte-character string to copy to *mbs1*.
- mbs2_byte_length*

is the integer number of bytes in the *mbs2* string. If *mbs2_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs2* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).
- char_limit*

is the maximum number of multibyte characters to read from *mbs2*.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsncpy()** function copies *mbs2* to the location that *mbs1* references. The function reads no more than *char_limit* characters from *mbs2* and writes them to *mbs1*. If *mbs1* and *mbs2* overlap, the results of this function are undefined.

If *mbs2_byte_length* is equal to `IFX_GL_NULL`, the function null-terminates the concatenated string. Any other value of *mbs2_byte_length* means that the function does not null-terminate the resulting concatenated string.

Return Values

- >=0 The number of bytes in the copied string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	The <i>mbs2_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0.
IFX_GL_EILSEQ	The <i>*mbs2</i> value contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscpy()**, **ifx_gl_wcscpy()**, and **ifx_gl_wcsncpy()**.

For more information, see [“String Copying” on page 2-33](#).

ifx_gl_mbsnext()

The **ifx_gl_mbsnext()** function returns the next multibyte character from a multibyte string.

Syntax

```
#include <ifxgls.h>
...
gl_mchar_t *ifx_gl_mbsnext(mb, mb_byte_limit)
    gl_mchar_t *mb;
    int mb_byte_limit;
```

- mb** is a pointer to the current multibyte character in the multibyte string.
- mb_byte_limit** is the integer number of bytes to read from *mb* to try to form a complete multibyte character. If *mb_byte_limit* is `IFX_GL_NO_LIMIT`, the function reads as many bytes as necessary from *mb* to form a complete character. For general information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsnext()** function returns a pointer to the next multibyte character after *mb* in a multibyte-character string. This function is typically used to transform the single-byte forward-traversal loop to a multibyte loop. For example, suppose you have the following single-byte loop:

```
for ( sb = sbs; *sb != '\0'; sb++ )
{
    /* Process single-byte character */
}
```

The following code fragment traverses the string with the **ifx_gl_mbsnext()** function instead of incrementing the pointer of the single-byte loop:

```
for ( mb = mbs; *mb != '\0'; )
{
    /* Process multibyte character; increment pointer through string */
    if ( (mb = ifx_gl_mbsnext(mb, IFX_GL_NO_LIMIT)) == NULL )
        /* Handle error */
}
```

Return Values

- gl_mchar_t* A pointer to the byte that immediately follows *mb*.
- NULL The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns NULL and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mblen()**, and **ifx_gl_mbsprev()**.

For more information, see [“String Traversal” on page 2-30](#).

ifx_gl_mbsntsbytes()

The **ifx_gl_mbsntsbytes()** function determines the number of bytes in a multibyte string. It ignores trailing spaces.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbsntsbytes(mbs, mbs_byte_length)
    gl_mchar_t *mbs;
    int mbs_byte_length;
```

mbs is a pointer to the multibyte-character string whose length the function determines.

mbs_byte_length is the integer number of bytes in the *mbs* string. If *mbs_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsntsbytes()** function returns the number of bytes in *mbs*, not including the trailing-space characters. The trailing-space characters are all characters that the blank character class of the current locale defines. For more information on the blank class, see [“Character Classification” on page 2-11](#).

Space characters that are embedded in the string at any place except the end of the string are included in the count. For example, the following call to **ifx_gl_mbsntsbytes()** returns a value of 10 (the number of bytes in the string “A¹A²B¹B²B³cdE¹”, where A¹A², B¹B²B³, and E¹ are multibyte characters of 2, 3, and 1 bytes, respectively, and s is a single-byte blank space):

```
ifx_gl_mbsntsbytes("A1A2B1B2B3cdsE1ssssss")
```

Return Values

- >=0 The number of bytes in the *mbs* string, not including any trailing space.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	The <i>mbs_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_EILSEQ	The <i>*mbs</i> value contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether the last character of <i>mbs</i> is a valid multibyte character because it would need to read more than <i>mbs_byte_length</i> bytes from <i>mbs</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsntslen()**, and **ifx_gl_wcsntslen()**.

For more information, see [“String-Length Determination”](#) on page 2-33.

ifx_gl_mbsntslen()

The **ifx_gl_mbsntslen()** function determines the number of characters in a multibyte string. It ignores trailing spaces.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbsntslen(mbs, mbs_byte_length)
    gl_mchar_t *mbs;
    int mbs_byte_length;
```

- mbs* is a pointer to the multibyte-character string whose length the function determines.
- mbs_byte_length* is the integer number of bytes in the *mbs* string. If *mbs_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsntslen()** function returns the number of characters in *mbs*, not including any trailing-space characters. The trailing-space characters are all characters that the blank character class of the current locale defines. For more information on the blank class, see [“Character Classification” on page 2-11](#).

Space characters that are embedded in the string at any place *except* the end of the string are included in the count. For example, the following call to **ifx_gl_mbsntslen()** returns a value of 7 (the number of bytes in the string “A¹A²B¹B²B³cdE¹”, where A¹A², B¹B²B³, and E¹ are multibyte characters of 2, 3, and 1 bytes, respectively, and s is a single-byte blank space):

```
ifx_gl_mbsntslen("A1A2B1B2B3cdsE1sssss")
```

Use the **ifx_gl_mbslen()** function if you want the length to include trailing white space.

Return Values

- >=0 The number of characters in the *mbs* string, not including any trailing space.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	The <i>mbs_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_EILSEQ	The <i>*mbs</i> value contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether the last character of <i>mbs</i> is a valid multibyte character because it would need to read more than <i>mbs_byte_length</i> bytes from <i>mbs</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbslen()**, **ifx_gl_mbsntslen()**, and **ifx_gl_wcsntslen()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_mbspbrk()

The **ifx_gl_mbspbrk()** function searches for any multibyte character from one multibyte string in another multibyte-character string.

Syntax

```
#include <ifxgls.h>
..
gl_mchar_t *ifx_gl_mbspbrk(mbs1, mbs1_byte_length, mbs2, mbs2_byte_length)
    gl_mchar_t *mbs1;
    int mbs1_byte_length;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
```

<i>mbs1</i>	is a pointer to the string of multibyte characters to search for the characters in <i>mbs2</i> .
<i>mbs1_byte_length</i>	is the integer number of bytes in the <i>mbs1</i> string. If <i>mbs1_byte_length</i> is the value IFX_GL_NULL, the function assumes that <i>mbs1</i> is a null-terminated string. For more general information about string lengths, see “Character-String Termination” on page 2-38 .
<i>mbs2</i>	is a pointer to the multibyte-character string whose characters the function searches for in <i>mbs1</i> .
<i>mbs2_byte_length</i>	is the integer number of bytes in the <i>mbs2</i> string. If <i>mbs2_byte_length</i> is the value IFX_GL_NULL, the function assumes that <i>mbs2</i> is a null-terminated string. For more general information about string lengths, see “Character-String Termination” on page 2-38 .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbspbrk()** function searches for the first occurrence of any multibyte character from the string *mbs2* in the multibyte string *mbs1*.

Return Values

<code>gl_mchar_t *</code>	A pointer to the first byte of the first occurrence in <i>mbs1</i> of any character from <i>mbs2</i> .
<code>NULL</code>	Either no character in <i>mbs2</i> was found in <i>mbs1</i> (the ifx_gl_lc_errno() error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
<code>IFX_GL_PARAMERR</code>	Either <i>mbs1_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0, or <i>mbs2_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0.
<code>IFX_GL_TERMMISM AT</code>	Either <i>mbs1_byte_length</i> is equal to <code>IFX_GL_NULL</code> , and <i>mbs2_byte_length</i> is greater than or equal to 0; or <i>mbs1_byte_length</i> is greater than or equal to 0, and <i>mbs2_byte_length</i> is equal to <code>IFX_GL_NULL</code> .
<code>IFX_GL_EILSEQ</code>	Either <i>mbs1</i> or <i>mbs2</i> contains an invalid multibyte character.
<code>IFX_GL_EINVAL</code>	Either the function cannot determine whether the last character of <i>mbs1</i> is a valid multibyte character because it would need to read more than <i>mbs1_byte_length</i> bytes from <i>mbs1</i> , or the function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbschr()**, **ifx_gl_mbsrchr()**, **ifx_gl_wcschr()**, **ifx_gl_wcsrchr()**, and **ifx_gl_wcspbrk()**.

For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_mbsprev()

The **ifx_gl_mbsprev()** function returns the previous multibyte character in a multibyte string.

Syntax

```
#include <ifxgls.h>
...
gl_mchar_t *ifx_gl_mbsprev(mbs_start, mb)
    gl_mchar_t *mbs_start;
    gl_mchar_t *mb;
```

mbs_start is a pointer to beginning of the multibyte string.

mb is a pointer to the current multibyte character in the multibyte string.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsprev()** function returns a pointer to the multibyte character before *mb* in a multibyte-character string. This function is typically used to transform the single-byte reverse-traversal loop to a multibyte loop. For example, suppose you have the following single-byte loop:

```
sb = sb0 + strlen(sb0);
for ( sb--; sb >= sb0; sb-- )
    /* Process single-byte character */
```

The following code fragment traverses the string with the **ifx_gl_mbsprev()** function instead of decrementing the pointer of the single-byte loop:

```
mb = mbs_start + strlen(mbs_start);
for ( mb = ifx_gl_mbsprev(mbs_start, mb) ; mb >= mbs_start;
      mb = ifx_gl_mbsprev(mbs_start, mb) )
    /* Process multibyte character */
```

Return Values

<code>gl_mchar_t *</code>	A pointer to the first byte of the multibyte character immediately before <i>mb</i> .
<code>NULL</code>	The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets the **`ifx_gl_lc_errno()`** error number to one of the following values.

Value	Description
<code>IFX_GL_EILSEQ</code>	The <i>*mb</i> character or a character between <i>mbs_start</i> and <i>mb</i> is not a valid multibyte character.
<code>IFX_GL_EINVPTR</code>	The <i>mb</i> value is less than or equal to <i>mbs_start</i> .
<code>IFX_GL_EINVAL</code>	The function cannot determine whether the character before <i>mb</i> is a valid multibyte character because it would need to read beyond <i>mb</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43.

Related Topics

See the descriptions of the following functions: **`ifx_gl_lc_errno()`**, **`ifx_gl_mblen()`**, and **`ifx_gl_mbsnext()`**.

For more information, see [“String Traversal”](#) on page 2-30.

ifx_gl_mbsrchr()

The **ifx_gl_mbsrchr()** function searches for the last occurrence of a character in a multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
gl_mchar_t *ifx_gl_mbsrchr(mbs, mbs_byte_length, mb, mb_byte_limit)
    gl_mchar_t *mbs;
    int mbs_byte_length;
    gl_mchar_t *mb;
    int mb_byte_limit;
```

<i>mbs</i>	is a pointer to the multibyte-character string in which the function searches for <i>mb</i> .
<i>mbs_byte_length</i>	is the integer number of bytes in the <i>mbs</i> string. If <i>mbs_byte_length</i> is the value IFX_GL_NULL, the function assumes that <i>mbs1</i> is a null-terminated string. For more general information about string lengths, see “Character-String Termination” on page 2-38 .
<i>mb</i>	is a pointer to the multibyte character to search for in <i>mbs</i> .
<i>mb_byte_limit</i>	is the number of bytes to read from the <i>mb</i> character to try to form a complete multibyte character. If <i>mb_byte_limit</i> is IFX_GL_NO_LIMIT, this function reads as many bytes as necessary from <i>mb</i> to form a complete character. For general information about <i>mb_byte_limit</i> , see “Multibyte-Character Termination” on page 2-39 .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsrchr()** function locates the last occurrence of *mb* in the multibyte string *mbs*.

Return Values

- gl_mchar_t *** A pointer to the last occurrence of *mb* in *mbs*.
- NULL** Either *mb* was not found in *mbs* (the **ifx_gl_lc_errno()** error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns **NULL** and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is an invalid multibyte character, or <i>mbs</i> contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> , or the function cannot determine whether the last character of <i>mbs</i> is a valid multibyte character because it would need to read more than <i>mbs_byte_length</i> bytes from <i>mbs</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .
IFX_GL_PARAMERR	The <i>mbs_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.

ifx_gl_mbsrchr()

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsrchr()**, **ifx_gl_wcschr()**, and **ifx_gl_wcsrchr()**.

For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_mbsspn()

The **ifx_gl_mbsspn()** function determines the length of a multibyte substring for a specified multibyte-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbsspn(mbs1, mbs1_byte_length, mbs2, mbs2_byte_length)
    gl_mchar_t *mbs1;
    int mbs1_byte_length;
    gl_mchar_t *mbs2;
    int mbs2_byte_length;
```

- mbs1*

is a pointer to the multibyte string to check for a substring of characters found in *mbs2*.
- mbs1_byte_length*

is the integer number of bytes in the *mbs1* string. If *mbs1_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs1* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).
- mbs2*

is a pointer to the multibyte-character substring whose characters the functions searches for in *mbs1*.
- mbs2_byte_length*

is the integer number of bytes in the *mbs2* string. If *mbs2_byte_length* is the value IFX_GL_NULL, the function assumes that *mbs2* is a null-terminated string. For more general information about string lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbsspn()** function returns the size of the initial substring of *mbs1* that contains only characters that match characters in *mbs2*. The initial substring begins at the first character of *mbs1*. Therefore, this size is the number of characters in *mbs1* before the first character that is not found in *mbs2*. For example, suppose you have the following two multibyte-characters strings, **mbs1** and **mbs2**:

```
mbs1 = "A1A2B1B2B3C1C2D1D2D3A1A2E1F1F2A1A2G1D1D2D3";
mbs2 = "B1B2B3D1D2D3A1A2C1C2";
```

With these multibyte strings, the following call to the **ifx_gl_mbsspn()** function returns 5:

```
ifx_gl_mbsspn(mbs1, bytelen1, mbs2, bytelen2)
```

The first five characters of **mbs1** are in **mbs2**. The sixth character in **mbs1** is E¹, which is *not* a character that matches one of the characters in **mbs2**.

Return Values

- >=0 The number of characters in *mbs1* before the first character for which *mbs1* and *mbs2* differ.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets **ifx_gl_lc_errno()** to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>mbs1_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0, or <i>mbs2_byte_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>mbs1_byte_length</i> is equal to IFX_GL_NULL, and <i>mbs2_byte_length</i> is greater than or equal to 0; or <i>mbs1_byte_length</i> is greater than or equal to 0, and <i>mbs2_byte_length</i> is equal to IFX_GL_NULL.
IFX_GL_EILSEQ	Either <i>mbs1</i> or <i>mbs2</i> contains an invalid multibyte character.
IFX_GL_EINVAL	Either the function cannot determine whether the last character of <i>mbs1</i> is a valid multibyte character because it would need to read more than <i>mbs1_byte_length</i> bytes from <i>mbs1</i> , or the function cannot determine whether the last character of <i>mbs2</i> is a valid multibyte character because it would need to read more than <i>mbs2_byte_length</i> bytes from <i>mbs2</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsspn()**, **ifx_gl_wcssp()**, and **ifx_gl_wcsspn()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_mbstowcs()

The **ifx_gl_mbstowcs()** function converts a multibyte character string to its wide-character representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbstowcs(wcs, mbs, mbs_byte_length, char_limit)
    gl_wchar_t *wcs;
    gl_mchar_t *mbs;
    int mbs_byte_length;
    int char_limit;
```

<i>wcs</i>	is a pointer to the wide-character string that contains the wide-character equivalent of <i>mbs</i> .
<i>mbs</i>	is a pointer to the multibyte-character string to convert to the <i>wcs</i> wide-character string.
<i>mbs_byte_length</i>	is the integer number of bytes in the <i>mbs</i> string. If <i>mbs_byte_length</i> is the value IFX_GL_NULL, the function assumes that <i>mbs</i> is a null-terminated string. For more general information about string lengths, see “Character-String Termination” on page 2-38 .
<i>char_limit</i>	is the maximum number of multibyte characters to read from <i>mbs</i> .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbstowcs()** function converts *mbs* into its wide-character representation and stores the result in the location that *wcs* references. The function reads no more than *char_limit* characters from *mbs* and writes them to *wcs*.

If *mbs_byte_length* is equal to `IFX_GL_NULL`, the function null-terminates the string in *wcs*. Any other value of *mbs_byte_length* means that the function does not null-terminate the resulting wide-character string.

Return Values

- >=0 The number of characters read from *mbs* and written to *wcs*, not counting any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	The <i>mbs_byte_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0.
IFX_GL_EILSEQ	The <i>*mbs</i> value contains an invalid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether the last character of <i>mbs</i> is a valid multibyte character because it would need to read more than <i>mbs_byte_length</i> bytes from <i>mbs</i> . For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

ifx_gl_mbstowcs()

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbtowc()**, and **ifx_gl_wcstombs()**.

ifx_gl_mbtowc()

The **ifx_gl_mbtowc()** function converts one multibyte character to its wide-character representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_mbtowc(wc, mb, mb_byte_limit)
    gl_wchar_t *wc;
    gl_mchar_t *mb;
    int mb_byte_limit;
```

<i>wc</i>	is a pointer to the wide character that contains the wide-character equivalent for <i>mb</i> .
<i>mb</i>	is a pointer to the multibyte character to convert to the <i>wc</i> wide character.
<i>mb_byte_limit</i>	is the integer number of bytes to read from <i>mb</i> to try to form a complete multibyte character. If <i>mb_byte_limit</i> is <code>IFX_GL_NO_LIMIT</code> , the function reads as many bytes as necessary from <i>mb</i> to form a complete character. For general information about <i>mb_byte_limit</i> , see “Multibyte-Character Termination” on page 2-39 .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_mbtowc()** function converts the multibyte character *mb* into its wide-character representation and stores the result in the wide character that *wc* references.

Return Values

- >=0 The number of bytes read from *mb*.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbstowcs()**, **ifx_gl_wcstombs()**, and **ifx_gl_wctomb()**.

ifx_gl_putmb()

The **ifx_gl_putmb()** function puts a single multibyte character in a user-defined location.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_putmb(mb, mb_byte_limit, funcp, v, bytes_put)
    gl_mchar_t *mb;
    int mb_byte_limit;
    int (*funcp)(gl_mchar_t byte, void *v);
    void *v;
    int *bytes_put;
```

- mb** is a pointer to the multibyte character whose bytes the *funcp* function writes to a specified location.
- mb_byte_limit** is the number of bytes in *mb* that the **ifx_gl_putmb()** function reads when it tries to form a complete multibyte character. If *mb_byte_limit* is IFX_GL_NO_LIMIT, this function reads as many bytes as necessary from *mb* to form a complete character. For general information about *mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).
- funcp** is a pointer to a function that you define to specify the location at which to write the multibyte character.
- v** is an argument to the user-defined function, which is a parameter to **ifx_gl_putmb()**. Use *v* to specify where the character should go; for example, stdout, stderr, a file, and so on.
- bytes_put** is a pointer to an integer that **ifx_gl_putmb()** sets to indicate the number of bytes that the *funcp* function has successfully written.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_putmb()** function calls a function that you define to put the bytes that form one multibyte character in a specified location. This multibyte character is read from the *mb* buffer.

The *funcp* argument is a pointer to a function that you must define as follows:

```
int funcp(gl_mchar_t byte, void *v)
```

On success, *funcp* must return 0. On failure, *funcp* must return -1. The number of bytes that *funcp* successfully puts is returned in *bytes_put* (even when *funcp* fails).

Return Values

- 0 The function was successful.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The <i>funcp</i> function cannot determine whether <i>mb</i> is a valid multibyte character because it would need to read more than <i>mb_byte_limit</i> bytes from <i>mb</i> . If <i>mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .
IFX_GL_EIO	The <i>funcp</i> function returned -1 when first called.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()** and **ifx_gl_getmb()**.

ifx_gl_tomlower()

The **ifx_gl_tomlower()** function converts an uppercase multibyte character to its lowercase equivalent.

Syntax

```
#include <ifxgls.h>
...
unsigned short ifx_gl_tomlower(dst_mb, src_mb, src_mb_byte_limit)
    gl_mchar_t *dst_mb;
    gl_mchar_t *src_mb;
    int src_mb_byte_limit;
```

- dst_mb*

is a pointer to the destination character, which holds the case conversion for the *src_mb* character.
- src_mb*

is a pointer to the source multibyte character that you want to convert to its lowercase equivalent.
- src_mb_byte_limit*

is the integer number of bytes to read from *src_mb* to try to form a complete multibyte character. If *src_mb_byte_limit* is IFX_GL_NO_LIMIT, this function reads as many bytes as necessary from *src_mb* to form a complete multibyte character. For general information about *src_mb_byte_limit*, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_tomlower()** function obtains the lowercase equivalent of the *src_mb* source multibyte character and stores it in the *dst_mb* destination buffer. If the locale does not define a lowercase equivalent for *src_mb*, **ifx_gl_tomlower()** copies *src_mb* to *dst_mb* unchanged.

For a multibyte-character string, the size of the lowercase multibyte string might not equal the size of the uppercase string. Therefore, to perform case conversion on multibyte characters, you must take the following special processing steps:

- Determine whether you need to allocate a separate destination buffer.
If a destination buffer is needed, determine its size
- Determine the number of bytes that have been read and written in the case-conversion process.

For more information on these steps, see [“Case Conversion for Multibyte Characters” on page 2-16](#).

The **ifx_gl_tolower()** function returns an unsigned short integer that encodes the number of bytes read from *src_mb* and the number of bytes written to *dst_mb*. The INFORMIX-GLS library provides the following macros to obtain this information from the return value.

INFORMIX-GLS Macro	Information Obtained
IFX_GL_CASE_CONV_SRC_BYTES()	The number of bytes read from the source string
IFX_GL_CASE_CONV_DST_BYTES()	The number of bytes written to the destination buffer

Return Values

- >0 An unsigned short integer that encodes the number of bytes read from *src_mb* and the number of bytes written to *dst_mb*.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*src_mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>src_mb</i> is a valid multibyte character because it would need to read more than <i>src_mb_byte_limit</i> bytes from <i>src_mb</i> . If <i>src_mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions:

ifx_gl_case_conv_outbuflen(), **ifx_gl_ismlower()**, **ifx_gl_ismupper()**, **ifx_gl_lc_errno()**, **ifx_gl_mb_loc_max()**, **ifx_gl_tomupper()**, and **ifx_gl_towlower()**.

For more information, see [“Case Conversion” on page 2-15](#).

ifx_gl_toupper()

The **ifx_gl_toupper()** function converts a lowercase multibyte character to its uppercase equivalent.

Syntax

```
#include <ifxgls.h>
...
unsigned short ifx_gl_toupper(dst_mb, src_mb, src_mb_byte_limit)
    gl_mchar_t *dst_mb;
    gl_mchar_t *src_mb;
    int src_mb_byte_limit;
```

- dst_mb

is a pointer to the destination character, which holds the case conversion for the src_mb character.
- src_mb

is a pointer to the source multibyte character that you want to convert to its uppercase equivalent.
- src_mb_byte_limit

is the integer number of bytes to read from src_mb to try to form a complete multibyte character. If src_mb_byte_limit is IFX_GL_NO_LIMIT, this function reads as many bytes as necessary from src_mb to form a complete multibyte character. For general information about src_mb_byte_limit, see [“Multibyte-Character Termination” on page 2-39](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_toupper()** function obtains the uppercase equivalent of the *src_mb* source multibyte character and stores it in the *dst_mb* destination buffer. If the locale does not define an uppercase equivalent for *src_mb*, **ifx_gl_toupper()** copies *src_mb* to *dst_mb* unchanged.

For a multibyte-character string, the size of the uppercase multibyte string might not equal the size of the lowercase string. Therefore, to perform case conversion on multibyte characters, you must take the following special processing steps:

- Determine whether you need to allocate a separate destination buffer.
If a destination buffer is needed, determine its size.
- Determine the number of bytes that have been read and written in the case-conversion process.

For more information on these steps, see [“Case Conversion for Multibyte Characters” on page 2-16](#).

The **ifx_gl_toupper()** function returns an unsigned short integer that encodes the number of bytes read from *src_mb* and the number of bytes written to *dst_mb*. The INFORMIX-GLS library provides the following macros to obtain this information from the return value.

INFORMIX-GLS Macro	Information Obtained
IFX_GL_CASE_CONV_SRC_BYTES()	The number of bytes read from the source string
IFX_GL_CASE_CONV_DST_BYTES()	The number of bytes written to the destination buffer

Return Values

- >0 An unsigned short integer that encodes the number of bytes read from *src_mb* and the number of bytes written to *dst_mb*.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does *not* return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_EILSEQ	The <i>*src_mb</i> value is not a valid multibyte character.
IFX_GL_EINVAL	The function cannot determine whether <i>src_mb</i> is a valid multibyte character because it would need to read more than <i>src_mb_byte_limit</i> bytes from <i>src_mb</i> . If <i>src_mb_byte_limit</i> is less than or equal to 0, this function always returns this error. For more information about this error, see “Keeping Multibyte Strings Consistent” on page 2-43 .

Related Topics

See the descriptions of the following functions:

ifx_gl_case_conv_outbuflen(), **ifx_gl_ismlower()**, **ifx_gl_ismupper()**, **ifx_gl_lc_errno()**, **ifx_gl_mb_loc_max()**, **ifx_gl_tomlower()**, and **ifx_gl_towupper()**.

For more information, see [“Case Conversion” on page 2-15](#).

ifx_gl_towlower()

The **ifx_gl_towlower()** function converts an uppercase wide character to its lowercase equivalent.

Syntax

```
#include <ifxgls.h>
...
unsigned short ifx_gl_towlower(src_wc)
    gl_mchar_t *src_wc;
```

src_wc is a pointer to the source wide character that you want to convert to its lowercase equivalent.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_towlower()** function returns the lowercase equivalent of the *src_wc* source wide character. If the locale does not define a lowercase equivalent for *src_wc*, **ifx_gl_towlower()** returns *src_wc* unchanged.

Return Values

- >0 An unsigned short integer that represents the lowercase equivalent of *src_wc*.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <i>*src_wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions:

ifx_gl_case_conv_outbuflen(), **ifx_gl_iswlower()**, **ifx_gl_iswupper()**, **ifx_gl_lc_errno()**, **ifx_gl_mb_loc_max()**, **ifx_gl_tomlower()**, and **ifx_gl_towupper()**.

For more information, see [“Case Conversion” on page 2-15](#).

ifx_gl_towupper()

The **ifx_gl_towupper()** function converts a lowercase wide character to its uppercase equivalent.

Syntax

```
#include <ifxgls.h>
...
unsigned short ifx_gl_towupper(src_wc)
    gl_mchar_t *src_wc;
```

src_wc is a pointer to the source wide character that you want to convert to its uppercase equivalent.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_towupper()** function returns the uppercase equivalent of the *src_wc* source wide character. If the locale does not define a uppercase equivalent for *src_wc*, **ifx_gl_towupper()** returns *src_wc* unchanged.

Return Values

- >0 An unsigned short integer that represents the uppercase equivalent of *src_wc*.
- 0 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

This function does not return a unique value to indicate an error. If an error occurred, the function returns 0 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The <i>*src_wc</i> value is not a valid wide character.

Related Topics

See the descriptions of the following functions:

ifx_gl_case_conv_outbuflen(), **ifx_gl_iswlower()**, **ifx_gl_iswupper()**, **ifx_gl_lc_errno()**, **ifx_gl_mb_loc_max()**, **ifx_gl_tomupper()**, and **ifx_gl_towlower()**.

For more information, see [“Case Conversion” on page 2-15](#).

ifx_gl_wcscat()

The **ifx_gl_wcscat()** function concatenates two wide-character strings.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcscat(wcs1, wcs1_char_length, wcs2, wcs2_char_length)
    gl_wchar_t *wcs1;
    int wcs1_char_length;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
```

- wcs1

is a pointer to the wide-character string to which the function concatenates wcs2.
- wcs1_char_length

is the integer number of characters in the wcs1 string. If wcs1_char_length is the value IFX_GL_NULL, the function assumes that wcs1 is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38.](#)
- wcs2

is a pointer to the wide-character string to concatenate onto wcs1.
- wcs2_char_length

is the integer number of characters in the wcs2 string. If wcs2_char_length is the value IFX_GL_NULL, the function assumes that wcs2 is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38.](#)

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcscat()** function appends a copy of wcs2 to the end of wcs1. If wcs1 and wcs2 overlap, the results of this function are undefined.

Return Values

- >=0 The number of characters in the concatenated string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>wcs1_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0, or <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>wcs1_char_length</i> is equal to IFX_GL_NULL, and <i>wcs2_char_length</i> is greater than or equal to 0; or <i>wcs1_char_length</i> is greater than or equal to 0, and <i>wcs2_char_length</i> is equal to IFX_GL_NULL.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscat()**, **ifx_gl_mbsncat()**, and **ifx_gl_wcsncat()**.

For more information, see [“Concatenation” on page 2-32](#).

ifx_gl_wcschr()

The **ifx_gl_wcschr()** function searches for the first occurrence of a character in a wide-character string.

Syntax

```
#include <ifxgls.h>
...
gl_wchar_t *ifx_gl_wcschr(wcs, wcs_char_length, wc)
gl_wchar_t *wcs;
int wcs_char_length;
gl_wchar_t wc;
```

wcs

is a pointer to the wide-character string in which the function searches for *wc*.

wcs_char_length

is the integer number of characters in the *wcs* string. If *wcs_char_length* is the value `IFX_GL_NULL`, the function assumes that *wcs* is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).

wc

is a pointer to the wide character to search for in *wcs*.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcschr()** function locates the first occurrence of *wc* in the wide-character string *wcs*.

Return Values

- gl_wchar_t * A pointer to the first occurrence of *wc* in *wcs*.
- NULL Either *wc* was not found in *wcs* (the **ifx_gl_lc_errno()** error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_PARAMERR	The <i>wcs_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbschr()**, **ifx_gl_mbsrchr()**, and **ifx_gl_wcsrchr()**.
For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_wcscoll()

The **ifx_gl_wcscoll()** function uses localized collation order to compare two wide-character strings.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcscoll(wcs1, wcs1_char_length, wcs2, wcs2_char_length)
    gl_wchar_t *wcs1;
    int wcs1_char_length;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
```

</

Locale Information

The LC_COLLATE category of the current locale affects the behavior of this function because it defines the localized collation order. For more information, see [“Character/String Comparison and Sorting” on page 2-35](#).

Return Values

- <0 The *wcs1* string is less than the *wcs2* string ($wcs1 < wcs2$); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.
- =0 The *wcs1* string is equal to the *wcs2* string ($wcs1 = wcs2$).
- >0 The *wcs1* string is greater than the *wcs2* string ($wcs1 > wcs2$).

Errors

This function does not return a unique value to indicate an error. If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>wcs1_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0, or <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>wcs1_char_length</i> is equal to IFX_GL_NULL, and <i>wcs2_char_length</i> is greater than or equal to 0; or <i>wcs1_char_length</i> is greater than or equal to 0, and <i>wcs2_char_length</i> is equal to IFX_GL_NULL.
IFX_GL_EILSEQ	Either <i>wcs1</i> or <i>wcs2</i> contains an invalid wide character.
IFX_GL_ENOMEM	Not enough memory is available to complete the operation.

ifx_gl_wcscoll()

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()** and **ifx_gl_mbscoll()**.

For more information, see [“Character/String Comparison and Sorting” on page 2-35](#).

ifx_gl_wcscpy()

The **ifx_gl_wcscpy()** function copies a wide-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcscpy(wcs1, wcs2, wcs2_char_length)
    gl_wchar_t *wcs1;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
```

- wcs1** is a pointer to the location where the function copies **wcs2**.
- wcs2** is a pointer to the wide-character string to copy to **wcs1**.
- wcs2_char_length** is the integer number of characters in the **wcs2** string. If **wcs2_char_length** is the value **IFX_GL_NULL**, the function assumes that **wcs2** is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcscpy()** function copies the wide-character string **wcs2** to the location that **wcs1** references. If **wcs1** and **wcs2** overlap, the results of this function are undefined.

Return Values

- >=0 The number of bytes in the copied string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_PARAMERR	The <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscpy()**, **ifx_gl_mbsncpy()**, and **ifx_gl_wcsncpy()**.

For more information, see [“String Copying” on page 2-33](#).

ifx_gl_wcscspn()

The **ifx_gl_wcscspn()** function determines the length of a complementary wide-character substring for a wide-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcscspn(wcs1, wcs1_char_length, wcs2, wcs2_char_length)
    gl_wchar_t *wcs1;
    int wcs1_char_length;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
```

- wcs1

is a pointer to the wide-character string to check for the complementary substring of characters in *wcs2*.
- wcs1_char_length

is the integer number of characters in the *wcs1* string. If *wcs1_char_length* is the value IFX_GL_NULL, the function assumes that *wcs1* is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- wcs2

is a pointer to the string of wide-characters to search in *wcs1*.
- wcs2_char_length

is the integer number of characters in the *wcs2* string. If *wcs2_char_length* is the value IFX_GL_NULL, the function assumes that *wcs2* is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcscspn()** function returns the size of the initial substring of *wcs1* that does not contain any characters that match characters in *wcs2*. The initial substring begins at the first character of *wcs1*. Therefore, this size is the number of characters in *wcs1* before the first character that the *wcs1* and *wcs2* strings have in common. The string of unmatched characters is called the *complementary string*.

For example, suppose you have the following two wide-character strings, **wcs1** and **wcs2**:

```
wcs1 = "A1A2B1B2C1C2D1D2A1A2E1E2F1F2A1A2G1G2D1D2";
wcs2 = "F1F2G1D1D2";
```

With these wide-character strings, the following call to the **ifx_gl_wcscspn()** function returns 3:

```
ifx_gl_wcscspn(wcs1, charlen1, wcs2, charlen2)
```

The first three characters of **wcs1** are *not* in **wcs2**. The fourth character in **wcs1** is D¹D², which is in **wcs2** also.

Return Values

- >=0 The number of characters in the initial substring of *wcs1* that consist entirely of wide characters not in the string *wcs2*.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>wcs1_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0; or <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>wcs1_char_length</i> is equal to IFX_GL_NULL, and <i>wcs2_char_length</i> is greater than or equal to 0; or <i>wcs1_char_length</i> is greater than or equal to 0, and <i>wcs2_char_length</i> is equal to IFX_GL_NULL.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscspn()**, **ifx_gl_mbsspn()**, and **ifx_gl_wcscspn()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_wcslen()

The **ifx_gl_wcslen()** function determines the number of characters in a wide-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcslen(wcs)
    gl_wchar_t *wcs;
```

wcs is a pointer to the wide-character string whose length the function determines.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcslen()** function computes the number of wide characters in the wide-character string to which wcs points, not including the null-terminating wide-character code. The length that **ifx_gl_wcslen()** returns includes any trailing white space. The trailing-space characters are all characters that the blank character class of the current locale defines. For more information on the blank class, see [“Character Classification” on page 2-11](#).

For example, the following call to **ifx_gl_wcslen()** return a value of 7 (where A¹A², B¹B², and C¹C² are wide characters of 2 bytes each, and s¹s² is the wide-character blank space):

```
ifx_gl_wcslen("A1A2B1B2s1s2C1C2s1s2s1s2")
```

Use the **ifx_gl_wcntslen()** function if you want the length without trailing white space.

Return Values

- >=0 The number of characters in the wcs string, not including any null terminator but including any trailing white space.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The *wcs value contains an invalid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbslen()**, **ifx_gl_mbsntsbytes()**, **ifx_gl_mbsntslen()**, and **ifx_gl_wcsntslen()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_wcsncat()

The **ifx_gl_wcsncat()** function concatenates a specified number of wide characters in one wide-character string to another.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcsncat(wcs1, wcs1_char_length, wcs2, wcs2_char_length, char_limit)
    gl_wchar_t *wcs1;
    int wcs1_char_length;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
    int char_limit;
```

- wcs1

is a pointer to the wide-character string to which the function concatenates *wcs2*.
- wcs1_char_length

is the integer number of characters in the *wcs1* string. If *wcs1_char_length* is the value `IFX_GL_NULL`, the function assumes that *wcs1* is a null-terminated string. For more general information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- wcs2

is a pointer to the wide-character string to concatenate to *wcs1*.
- wcs2_char_length

is the integer number of characters in the *wcs2* string. If *wcs2_char_length* is the value `IFX_GL_NULL`, the function assumes that *wcs2* is a null-terminated string. For more general information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- char_limit

is the maximum number of wide characters to read from *wcs2*.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcsncat()** function appends *wcs2* to the end of *wcs1*. If *wcs1* and *wcs2* overlap, the results of this function are undefined. The function reads no more than *char_limit* characters from *wcs2* and writes them to *wcs1*.

Return Values

- >=0 The number of bytes in the concatenated string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_PARAMERR	Either <i>wcs1_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0, or <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>wcs1_char_length</i> is equal to IFX_GL_NULL, and <i>wcs2_char_length</i> is greater than or equal to 0; or <i>wcs1_char_length</i> is greater than or equal to 0, and <i>wcs2_char_length</i> is equal to IFX_GL_NULL.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscat()**, **ifx_gl_mbsncat()**, and **ifx_gl_wcscat()**.

For more information, see [“Concatenation” on page 2-32](#).

ifx_gl_wcsncpy()

The **ifx_gl_wcsncpy()** function copies a specified number of wide characters from a wide-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcsncpy(wcs1, wcs2, wcs2_char_length, char_limit)
    gl_wchar_t *wcs1;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
    int char_limit;
```

- wcs1

is a pointer to the location where the function copies *wcs2*.
- wcs2

is a pointer to the wide-character string to copy to *wcs1*.
- wcs2_char_length

is the integer number of characters in the *wcs2* string. If *wcs2_char_length* is the value `IFX_GL_NULL`, the function assumes that *wcs2* is a null-terminated string. For more general information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- char_limit

is the maximum number of wide characters to read from *wcs2*.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcsncpy()** function copies *wcs2* to the location that *wcs1* references. The function reads no more than *char_limit* characters from *wcs2* and writes them to *wcs1*. If *wcs1* and *wcs2* overlap, the results of this function are undefined.

Return Values

- >=0 The number of characters in the copied string, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_PARAMERR	The <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscpy()**, **ifx_gl_mbsncpy()**, and **ifx_gl_wcscpy()**.

For more information, see [“String Copying” on page 2-33](#).

ifx_gl_wcsntslen()

The **ifx_gl_wcsntslen()** function determines the number of characters in a wide-character string. It ignores trailing spaces.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcsntslen(wcs, wcs_char_length)
    gl_wchar_t *wcs;
    int wcs_char_length;
```

- wcs** is a pointer to the wide-character string whose length the function determines.
- wcs_char_length** is the integer number of characters in the *wcs* string. If *wcs_char_length* is the value IFX_GL_NULL, the function assumes that *wcs* is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcsntslen()** function returns the number of characters in *wcs*, not including the trailing-space characters. The trailing-space characters are all characters that the blank character class of the current locale defines. For more information on the blank class, see [“Character Classification” on page 2-11](#).

Space characters that are embedded in the string at any place except the end of the string are included in the count. For example, the following call to **ifx_gl_wcsntslen()** returns a value of 4 (the number of wide characters in the string “A¹A²B¹B²s¹s²C¹C²”, where A¹A², B¹B², and C¹C² are wide characters of 2 bytes each, and s¹s² is the wide-character blank space):

```
ifx_gl_wcsntslen("A1A2B1B2s1s2C1C2s1s2s1s2s1s2")
```

Use the **ifx_gl_wcslen()** function if you want the length to include trailing white space.

Return Values

- >=0 The number of characters in the wcs string, not including any trailing space.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_PARAMERR	The <i>wcs_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbsntsbytes()**, **ifx_gl_mbsntslen()**, and **ifx_gl_wcslen()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_wcsprk()

The **ifx_gl_wcsprk()** function searches for any wide character from one wide-character string in another wide-character string.

Syntax

```
#include <ifxgls.h>
...
gl_wchar_t *ifx_gl_wcsprk(wcs1, wcs1_char_length, wcs2, wcs2_char_length)
    gl_wchar_t *wcs1;
    int wcs1_char_length;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
```

- wcs1

is a pointer to the wide-character string to search for any character in *wcs2*.
- wcs1_char_length

is the integer number of characters in the *wcs1* string. If *wcs1_char_length* is the value `IFX_GL_NULL`, the function assumes that *wcs1* is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- wcs2

is a pointer to the wide-character string whose characters the function searches for in *wcs1*.
- wcs2_char_length

is the integer number of characters in the *wcs2* string. If *wcs2_char_length* is the value `IFX_GL_NULL`, the function assumes that *wcs2* is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcsprbk()** function searches for the first occurrence of any wide character from the string *wcs2* in the wide-character string *wcs1*.

Return Values

- gl_wchar_t *** A pointer to the first byte of the first occurrence in *wcs1* of any character from *wcs2*.
- NULL** Either no character in *wcs2* was found in *wcs1* (the **ifx_gl_lc_errno()** error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns **NULL** and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>wcs1_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0, or <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>wcs1_char_length</i> is equal to IFX_GL_NULL, and <i>wcs2_char_length</i> is greater than or equal to 0; or <i>wcs1_char_length</i> is greater than or equal to 0, and <i>wcs2_char_length</i> is equal to IFX_GL_NULL.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbschr()**, **ifx_gl_mbspbrk()**, **ifx_gl_mbsrchr()**, **ifx_gl_wcschr()**, and **ifx_gl_wcsrchr()**.

For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_wcsrchr()

The **ifx_gl_wcsrchr()** function searches for the last occurrence of a character in a wide-character string.

Syntax

```
#include <ifxgls.h>
...
gl_wchar_t *ifx_gl_wcsrchr(wcs, wcs_char_length, wc)
    gl_wchar_t *wcs;
    int wcs_char_length;
    gl_wchar_t wc;
```

- wcs** is a pointer to the wide-character string in which the function searches for *wc*.
- wcs_char_length** is the integer number of characters in the *wcs* string. If *wcs_char_length* is the value *IFX_GL_NULL*, the function assumes that *wcs* is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- wc** is a pointer to the wide character to search for in *wcs*.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcsrchr()** function locates the last occurrence of *wc* in the wide character string *wcs*.

Return Values

gl_wchar_t *	A pointer to the last occurrence of <i>wc</i> in <i>wcs</i> .
NULL	Either <i>wc</i> was not found in <i>wcs</i> (the ifx_gl_lc_errno() error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_PARAMERR	The <i>wcs_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbschr()**, **ifx_gl_mbsrchr()**, and **ifx_gl_wcschr()**.

For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_wcsspncpy()

The **ifx_gl_wcsspncpy()** function determines the length of a wide-character substring for a specified wide-character string.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcsspncpy(wcs1, wcs1_char_length, wcs2, wcs2_char_length)
    l_wchar_t *wcs1;
    int wcs1_char_length;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
```

- wcs1

is a pointer to the wide-character string to check for a substring of characters found in wcs2.
- wcs1_char_length

is the integer number of characters in the wcs1 string. If wcs1_char_length is the value IFX_GL_NULL, the function assumes that wcs1 is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- wcs2

is a pointer to the wide-character string whose characters the function searches for in wcs1.
- wcs2_char_length

is the integer number of characters in the wcs2 string. If wcs2_char_length is the value IFX_GL_NULL, the function assumes that wcs2 is a null-terminated string. For more information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcsspnl()** function returns the size of the initial substring of *wcs1* that contains only characters that match characters in *wcs2*. The initial substring begins at the first character of *wcs1*. Therefore, this size is the number of characters in *wcs1* before the first character that is not found in *wcs2*.

For example, suppose you have the following two wide-character strings, **wcs1** and **wcs2**:

```
wcs1 = "A1A2B1B2C1C2D1D2A1A2E1E2F1F2A1A2G1G2D1D2";
wcs2 = "B1B2D1D2A1A2C1C2";
```

With these multibyte strings, the following call to the **ifx_gl_wcsspnl()** function returns 5:

```
ifx_gl_wcsspnl(wcs1, charlen1, wcs2, charlen2)
```

The first five characters of **wcs1** are in **wcs2**. The sixth character in **wcs1** is E¹E², which is not a character that matches one of the characters in **wcs2**.

Return Values

- >=0 The number of characters in the initial substring of *wcs1* that consist entirely of wide characters in the string *wcs2*.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	Either <i>wcs1_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0, or <i>wcs2_char_length</i> is not equal to IFX_GL_NULL and is not greater than or equal to 0.
IFX_GL_TERMMISMAT	Either <i>wcs1_char_length</i> is equal to IFX_GL_NULL, and <i>wcs2_char_length</i> is greater than or equal to 0; or <i>wcs1_char_length</i> is greater than or equal to 0, and <i>wcs2_char_length</i> is equal to IFX_GL_NULL.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbscspn()**, **ifx_gl_mbssp()**, and **ifx_gl_wcscspn()**.

For more information, see [“String-Length Determination” on page 2-33](#).

ifx_gl_wcstombs()

The **ifx_gl_wcstombs()** function converts a wide-character string to its multibyte representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wcstombs(mbs, wcs, wcs_char_length, mbs_byte_limit)
    gl_mchar_t *mbs;
    gl_wchar_t *wcs;
    int wcs_char_length;
    int mbs_byte_limit;
```

- mbs*

is a pointer to the multibyte-character string that contains the multibyte equivalent for *wcs*.
- wcs*

is a pointer to the wide-character string to convert to the *mbs* multibyte string.
- wcs_char_length*

is the integer number of characters in the *wcs* string. If *wcs_char_length* is the value IFX_GL_NULL, the function assumes that *wcs* is a null-terminated string. For more general information about wide-character lengths, see [“Character-String Termination” on page 2-38](#).
- mbs_byte_limit*

is the integer number of bytes to read from *mbs*.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcstombs()** function converts the wide-character string *wcs* to its multibyte representation and stores the result in the location that *mbs* references.

The function writes at most *mbs_byte_limit* bytes to *mbs*. If a particular character would cause more than *mbs_byte_limit* bytes to be written to *mbs*, no part of that character is written to *mbs*. In this case, fewer than *mbs_byte_limit* bytes are written to *mbs*, but *mbs* is still considered full.

If *wcs_char_length* is equal to `IFX_GL_NULL`, the function null-terminates *mbs*. Any other value of *wcs_char_length* means that the function does not null-terminate the resulting string.

Return Values

- >=0 The number of characters that were written to *mbs*, not including any null terminator.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
IFX_GL_PARAMERR	The <i>wcs_char_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0.
IFX_GL_EILSEQ	The <i>*wcs</i> value contains an invalid wide character.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbstowcs()**, **ifx_gl_mbtowc()**, and **ifx_gl_wctomb()**.

ifx_gl_wcswcs()

The **ifx_gl_wcswcs()** function searches for a specified substring within a wide-character string.

Syntax

```
#include <ifxgls.h>
...
gl_wchar_t *ifx_gl_wcswcs(wcs1, wcs1_char_length, wcs2, wcs2_char_length)
    gl_wchar_t *wcs1;
    int wcs1_char_length;
    gl_wchar_t *wcs2;
    int wcs2_char_length;
```

<i>wcs1</i>	is a pointer to the wide-character string to search for the substring <i>wcs2</i> .
<i>wcs1_char_length</i>	is the integer number of characters in the <i>wcs1</i> string. If <i>wcs1_char_length</i> is the value <code>IFX_GL_NULL</code> , the function assumes that <i>wcs1</i> is a null-terminated string. For more information about wide-character lengths, see “Character-String Termination” on page 2-38 .
<i>wcs2</i>	is a pointer to the wide-character substring to search for in <i>wcs1</i> .
<i>wcs2_char_length</i>	is the integer number of characters in the <i>wcs2</i> string. If <i>wcs2_char_length</i> is the value <code>IFX_GL_NULL</code> , the function assumes that <i>wcs2</i> is a null-terminated string. For more information about wide-character lengths, see “Character-String Termination” on page 2-38 .

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wcswcs()** function searches for the first occurrence of the wide-character string *wcs2* in the wide-character string *wcs1*.

Return Values

<code>gl_wchar_t *</code>	A pointer to the first byte of the first occurrence of <i>wcs2</i> in <i>wcs1</i> .
<code>NULL</code>	Either <i>wcs2</i> was not found in <i>wcs1</i> (the ifx_gl_lc_errno() error number is set to 0); or the function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns `NULL` and sets the **ifx_gl_lc_errno()** error number to one of the following values.

Value	Description
<code>IFX_GL_PARAMERR</code>	Either <i>wcs1_char_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0, or <i>wcs2_char_length</i> is not equal to <code>IFX_GL_NULL</code> and is not greater than or equal to 0.
<code>IFX_GL_TERMMISMAT</code>	Either <i>wcs1_char_length</i> is equal to <code>IFX_GL_NULL</code> , and <i>wcs2_char_length</i> is greater than or equal to 0; or <i>wcs1_char_length</i> is greater than or equal to 0, and <i>wcs2_char_length</i> is equal to <code>IFX_GL_NULL</code> .

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mbschr()**, **ifx_gl_mbsmbs()**, **ifx_gl_mbsrchr()**, **ifx_gl_wcschr()**, and **ifx_gl_wcsrchr()**.

For more information, see [“Character Searching” on page 2-34](#).

ifx_gl_wctomb()

The **ifx_gl_wctomb()** function converts one wide character to its multibyte representation.

Syntax

```
#include <ifxgls.h>
...
int ifx_gl_wctomb(mb, wc)
    gl_mchar_t *mb;
    gl_wchar_t wc;
```

- mb* is a pointer to the multibyte character that contains the multibyte equivalent for *wc*.
- wc* is a pointer to the wide character to convert to the *mb* multibyte character.

Valid in Client Application	Valid in DataBlade UDR
Yes	Yes

Usage

The **ifx_gl_wctomb()** function converts the wide character in *wc* to its multibyte representation and stores the result in consecutive bytes that start at *mb*. This function assumes that *mb* contains enough space to hold the multibyte representation of *wc*. You can use either of the following methods to determine the number of bytes that will be written to *mb*:

- The function **ifx_gl_mb_loc_max()** calculates the maximum number of bytes in the multibyte representation for any wide character for the *current* locale.
- The macro **IFX_GL_MB_MAX** is the maximum number of bytes in the multibyte representation for any wide character in *any* locale.
The value is always greater than or equal to the value that **ifx_gl_mb_loc_max()** returns.

Of the two options, the macro **IFX_GL_MB_MAX** is faster and can initialize static buffers. The function **ifx_gl_mb_loc_max()** is slower but more precise.

Return Values

- >=0 The number of bytes written to *mb*.
- 1 The function was not successful, and the error number is set to indicate the cause. See the Errors section.

Errors

If an error occurred, this function returns -1 and sets the **ifx_gl_lc_errno()** error number to the following value.

Value	Description
IFX_GL_EILSEQ	The *wc value is not a valid wide character. In this case, some bytes might be written to <i>mb</i> , but the contents of <i>mb</i> are undefined.

Related Topics

See the descriptions of the following functions: **ifx_gl_lc_errno()**, **ifx_gl_mb_loc_max()**, **ifx_gl_mbstowcs()**, **ifx_gl_mbtowc()**, and **ifx_gl_wcstombs()**.

List of INFORMIX-GLS Error Numbers

This appendix provides a list of the error-number constants that the `ifx_gl_lc_errno()` function can retrieve. It also provides a list of the INFORMIX-GLS functions that might set a particular error number.



Important: The INFORMIX-GLS error-number constants are defined in the `gls.h` header file. For a complete list of error-number constants, refer to the `gls.h` header file.

Error-Number Constant	Description	INFORMIX-GLS Functions
IFX_GL_NOERRNO	No error has occurred.	Internal use only
IFX_GL_EILSEQ	Character sequence in a multibyte or wide character is invalid.	<code>ifx_gl_cv_mconv()</code> , <code>ifx_gl_getmb()</code> , <code>ifx_gl_ismalnum()</code> , <code>ifx_gl_ismalpha()</code> , <code>ifx_gl_ismblank()</code> , <code>ifx_gl_ismcntrl()</code> , <code>ifx_gl_ismdigit()</code> , <code>ifx_gl_ismgraph()</code> , <code>ifx_gl_ismlower()</code> , <code>ifx_gl_ismprint()</code> , <code>ifx_gl_ismpunct()</code> , <code>ifx_gl_ismspace()</code> , <code>ifx_gl_ismupper()</code> , <code>ifx_gl_ismxdigit()</code> , <code>ifx_gl_iswalnum()</code> , <code>ifx_gl_iswalpha()</code> , <code>ifx_gl_iswblank()</code> , <code>ifx_gl_iswcntrl()</code> , <code>ifx_gl_iswdigit()</code> , <code>ifx_gl_iswgraph()</code> , <code>ifx_gl_iswlower()</code> , <code>ifx_gl_iswprint()</code> , <code>ifx_gl_iswpunct()</code> , <code>ifx_gl_iswspace()</code> , <code>ifx_gl_iswupper()</code> , <code>ifx_gl_iswxdigit()</code> , <code>ifx_gl_mblen()</code> , <code>ifx_gl_mbscat()</code> , <code>ifx_gl_mbschr()</code> , <code>ifx_gl_mbscoll()</code> , <code>ifx_gl_mbscpy()</code> , <code>ifx_gl_mbscspn()</code> , <code>ifx_gl_mbslen()</code> , <code>ifx_gl_mbsmbs()</code> , <code>ifx_gl_mbsncat()</code> , <code>ifx_gl_mbsncpy()</code> , <code>ifx_gl_mbsnext()</code> , <code>ifx_gl_mbsntsbytes()</code> , <code>ifx_gl_mbsntslenn()</code> , <code>ifx_gl_mbspbrk()</code> , <code>ifx_gl_mbsprev()</code> , <code>ifx_gl_mbsrchr()</code> , <code>ifx_gl_mbsspn()</code> , <code>ifx_gl_mbstowcs()</code> , <code>ifx_gl_mbtowc()</code> , <code>ifx_gl_putmb()</code> , <code>ifx_gl_tomlower()</code> , <code>ifx_gl_tomupper()</code> , <code>ifx_gl_towlower()</code> , <code>ifx_gl_towupper()</code> , <code>ifx_gl_wcscoll()</code> , <code>ifx_gl_wcslen()</code> , <code>ifx_gl_wcstombs()</code> , <code>ifx_gl_wctomb()</code>
IFX_GL_ENULLPTR	Function received a NULL pointer to a function.	Internal use only
IFX_GL_ENOMEM	Function encountered a memory-allocation failure.	<code>ifx_gl_init()</code> , <code>ifx_gl_mbscoll()</code> , <code>ifx_gl_wcscoll()</code>
IFX_GL_EINDEXRANGE	Index is out of bounds.	Internal use only
IFX_GL_EINVPTR	An end pointer is less than a begin pointer.	<code>ifx_gl_mbsprev()</code> , <code>ifx_gl_putmb()</code>

(1 of 7)

Error-Number Constant	Description	INFORMIX-GLS Functions
IFX_GL_ERANGE	The base of a number is out of range.	Internal use only
IFX_GL_EINVAL	Wide-character or multibyte-character string is invalid.	ifx_gl_conv_needed(), ifx_gl_cv_mconv(), ifx_gl_getmb(), ifx_gl_ismalnum(), ifx_gl_ismalpha(), ifx_gl_isblank(), ifx_gl_ismcntrl(), ifx_gl_ismdigit(), ifx_gl_ismgraph(), ifx_gl_ismlower(), ifx_gl_ismprint(), ifx_gl_ismpunct(), ifx_gl_ismspace(), ifx_gl_ismupper(), ifx_gl_ismxdigit(), ifx_gl_mblen(), ifx_gl_mbscat(), ifx_gl_mbschr(), ifx_gl_mbscoll(), ifx_gl_mbscpy(), ifx_gl_mbscspn(), ifx_gl_mbslen(), ifx_gl_mbsmbs(), ifx_gl_mbsncat(), ifx_gl_mbsncpy(), ifx_gl_mbsnext(), ifx_gl_mbsntsbytes(), ifx_gl_mbsntslens(), ifx_gl_mbspbrk(), ifx_gl_mbspPrev(), ifx_gl_mbsrchr(), ifx_gl_mbsspn(), ifx_gl_mbstowcs(), ifx_gl_mbtowc(), ifx_gl_putmb(), ifx_gl_tomlower(), ifx_gl_tomupper()
IFX_GL_FILEERR	Input file could not be read.	ifx_gl_cv_mconv(), ifx_gl_cv_outbuflen(), ifx_gl_cv_sb2sb_table(), ifx_gl_init()
IFX_GL_PARAMERR	Parameter is out of bounds.	ifx_gl_convert_number(), ifx_gl_format_number(), ifx_gl_init(), ifx_gl_mbscat(), ifx_gl_mbschr(), ifx_gl_mbscoll(), ifx_gl_mbscpy(), ifx_gl_mbscspn(), ifx_gl_mbslen(), ifx_gl_mbsmbs(), ifx_gl_mbsncat(), ifx_gl_mbsncpy(), ifx_gl_mbsntsbytes(), ifx_gl_mbsntslens(), ifx_gl_mbspbrk(), ifx_gl_mbsrchr(), ifx_gl_mbsspn(), ifx_gl_mbstowcs(), ifx_gl_wcscat(), ifx_gl_wcschr(), ifx_gl_wscoll(), ifx_gl_wscpy(), ifx_gl_wscspn(), ifx_gl_wcsncat(), ifx_gl_wcsncpy(), ifx_gl_wcsntslens(), ifx_gl_wcspbrk(), ifx_gl_wcsrchr(), ifx_gl_wcspn(), ifx_gl_wcstombs(), ifx_gl_wcswcs()

(2 of 7)

Error-Number Constant	Description	INFORMIX-GLS Functions
IFX_GL_CATASTROPHE	Function encountered an internal error. Result is undefined result.	Internal use only
IFX_GL_BADFILEFORM	File format was invalid.	ifx_gl_conv_needed(), ifx_gl_init()
IFX_GL_INVALIDLOC	Locale code sets are inconsistent.	ifx_gl_init()
IFX_GL_EIO	Function encountered an I/O error.	Internal use only
IFX_GL_E2BIG	Operation would overflow a buffer.	ifx_gl_convert_date(), ifx_gl_cv_mconv(), ifx_gl_format_date(), ifx_gl_format_datetime(), ifx_gl_format_money(), ifx_gl_format_number()
IFX_GL_EBADF	Function received a bad handle.	ifx_gl_conv_needed(), ifx_gl_convert_date(), ifx_gl_convert_datetime(), ifx_gl_format_date(), ifx_gl_format_datetime()
IFX_GL_EOF	Function encountered an end-of-file on input stream.	ifx_gl_getmb()
IFX_GL_EUNKNOWN	An unknown system error has occurred.	Internal use only
IFX_GL_UNLOADED CAT	Function cannot copy from an unloaded category.	Internal use only
IFX_GL_LOADED CAT	Function cannot copy into a loaded category.	Internal use only
IFX_GL_ENOSYS	A feature is not supported.	ifx_gl_format_money()
IFX_GL_ELOCTOOWIDE	The current locale has characters that are too wide for this version of the INFORMIX-GLS library.	ifx_gl_init()
IFX_GL_INVALIDFMT	A formatted argument string is invalid.	ifx_gl_convert_date(), ifx_gl_convert_money(), ifx_gl_convert_number(), ifx_gl_format_number()

(3 of 7)

Error-Number Constant	Description	INFORMIX-GLS Functions
IFX_GL_EFRACRANGE	Fraction of second is out of bounds.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_ESECONDRANGE	Second is out of bounds.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_EMINUTERANGE	Minute is out of bounds.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_EHOURRANGE	Hour is out of bounds.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_EDAYRANGE	Day number is out of bounds.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_EWKDAYRANGE	Weekday number is out of bounds.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_EYDAYRANGE	Year-day number is out of bounds.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GLEMONTHRANGE	Month number is out of bounds.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_EYEARRANGE	Year number is out of bounds.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_EERAOFFRANGE	Era offset is out of bounds.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADFRAC	Fraction could not be scanned.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>

(4 of 7)

Error-Number Constant	Description	INFORMIX-GLS Functions
IFX_GL_BADSECOND	Second could not be scanned.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADMINUTE	Minute could not be scanned.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADHOUR	Hour could not be scanned.	<code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADDAY	Month-day could not be scanned.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADWKDAY	Weekday could not be scanned.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADYDAY	Year-day could not be scanned.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADMONTH	Month could not be scanned.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADYEAR	Year could not be scanned.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADERANAME	Era name is invalid.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>
IFX_GL_BADERAOFFSET	Era offset is invalid.	<code>ifx_gl_convert_date()</code> , <code>ifx_gl_convert_datetime()</code> , <code>ifx_gl_format_date()</code> , <code>ifx_gl_format_datetime()</code>

(5 of 7)

Error-Number Constant	Description	INFORMIX-GLS Functions
IFX_GL_BADFMTMOD	Format modifier is invalid.	ifx_gl_convert_date()
IFX_GL_BADFMTWP	Field width or precision is invalid.	ifx_gl_convert_date()
IFX_GL_BADINPUT	Input string does not match format string.	ifx_gl_convert_date()
IFX_GL_NOPOINT	Input is missing a decimal separator.	ifx_gl_convert_date()
IFX_GL_BADMONTHSTR	Month string could not be scanned	ifx_gl_convert_date()
IFX_GL_BADERASPEC	Function could not load era from locale.	ifx_gl_convert_date() ,
IFX_GL_BADCALENDAR	LC_TIME category contains an unsupported calendar.	Internal use only
IFX_GL_BADOBJVER	The lc (locale), cm (code set), or cv (code-set conversion) object is the wrong version.	ifx_gl_init()
IFX_GL_BADALTDAT	Function could not convert %Z information.	Internal use only
IFX_GL_NOSYMMAP	Character / symbol is not in charmap.	Internal use only
IFX_GL_BADSYM	Symbolic character name is invalid.	Internal use only
IFX_GL_ELOCLOAD	Function could not load locale.	Internal use only

(6 of 7)

Error-Number Constant	Description	INFORMIX-GLS Functions
IFX_GL_TERMMISMAT	String-termination parameters do not match.	ifx_gl_mbscat(), ifx_gl_mbscoll(), ifx_gl_mbscspn(), ifx_gl_mbsmbs(), ifx_gl_mbsncat(), ifx_gl_mbspbrk(), ifx_gl_mbssp(), ifx_gl_wcscat(), ifx_gl_wcscoll(), ifx_gl_wscspn(), ifx_gl_wcsncat(), ifx_gl_wcspbrk(), ifx_gl_wcssp(), ifx_gl_wcswcs()
IFX_GL_NOCTYPE	GL_CTYPE is not loaded	Internal use only
IFX_GL_LOCMISMAT	Loaded code sets are not same.	Internal use only

(7 of 7)

Index

A

Alnum class 2-12, 2-13, 4-91
 Alpha class 2-11, 2-13, 4-91, 4-93,
 4-118, 4-120
 Alphabetic character
 case ignored 2-13, 4-91, 4-93,
 4-118, 4-120
 definition of 2-11
 lowercase 2-11, 2-13, 4-104, 4-130
 uppercase 2-12, 2-13, 4-113, 4-138
 ANSI compliance
 level Intro-15
 Array size
 multibyte-character string 2-41
 wide-character string 2-42
 ASCII code set 2-5
 Asian character 1-21

B

Blank character 2-12, 4-77
 See also Horizontal white space.
 Blank class 4-96, 4-122, 4-160
 defined 2-12
 testing for 2-13
 Boldface type Intro-6
 Buffer-size estimate
 case conversion 2-17, 2-24,
 4-11, 4-49
 code-set conversion 2-24,
 2-41, 4-49

C

Carriage-return character 2-12
 See also Vertical white space.
 Case conversion
 buffer-size estimate 2-17
 description of 2-15
 INFORMIX-GLS functions for 4-8
 lowercase to uppercase 4-199,
 4-204
 optimizing 2-21
 uppercase to lowercase 4-196,
 4-202
 CHAR data type
 See Character; Character string.
 Character
 7-bit 2-5
 8-bit 2-5
 alphabetic. *See* Alphabetic
 character.
 blank. *See* Blank character.
 case conversion 2-15
 classifying. *See* Character
 classification.
 comparing 2-35
 control. *See* Control character.
 digit. *See* Digit character.
 graphical. *See* Graphical character.
 hexadecimal digit. *See*
 Hexadecimal-digit character.
 lowercase. *See* Lowercase
 character.
 multibyte. *See* Multibyte
 character.
 operations on 2-10
 printable. *See* Printable character.

- punctuation. *See* Punctuation character.
- searching for 2-34
- single-byte. *See* Single-byte character.
- sorting 2-35
- space. *See* Space character.
- types of 2-4
- uppercase. *See* Uppercase character.
- visible 4-102, 4-128
- wide. *See* Wide character.
- Character classification
 - alnum class 2-12, 4-91
 - alpha class 2-11, 4-91, 4-93, 4-118, 4-120
 - blank class 2-12, 4-96, 4-122
 - cntrl class 2-12, 4-98, 4-124
 - description of 2-11
 - digit class 2-12, 4-91, 4-100, 4-118, 4-126
 - error-checking examples 1-20
 - graph class 2-13, 4-102, 4-128
 - INFORMIX-GLS functions for 4-7
 - lower class 2-11, 4-104, 4-130
 - print class 2-13, 4-106, 4-132
 - punct class 2-13, 4-108, 4-134
 - space class 2-12, 4-110, 4-136
 - testing for 2-13
 - upper class 2-12, 4-113, 4-138
 - xdigit class 2-12, 4-115, 4-140
- Character operation
 - case conversion 2-15
 - character classification 2-11
 - code-set conversion 2-22
 - INFORMIX-GLS functions for 2-10
- Character processing 2-3, 4-6
- Character string
 - casting to multibyte string 2-7
 - comparing 2-35
 - complementary 4-158, 4-216
 - concatenating 2-32
 - conversion of 3-10
 - copying 2-33
 - date 3-6
 - date/time 3-6
 - deallocating 2-43
 - formatting of 3-11
 - length of 2-33
 - length-terminated 2-38
 - memory management 2-41, 2-42
 - money 3-5
 - null-terminated 2-38
 - number 3-4
 - operations on 2-30
 - processing of 2-32
 - searching 2-34
 - sorting 2-35
 - string traversal 2-30
 - termination of 2-38
 - See also* Multibyte-character string; Wide-character string.
- Charmap 12
- Client application
 - See* ESQ / C client application; DataBlade client application.
- Client locale 1-10, 1-11, 1-18
- CLIENT_LOCALE environment variable 1-11
- Cntrl class 2-12, 2-14, 4-98, 4-124
- Code point 2-4, 2-5, 2-35
- Code set 2-4, 2-8
 - ASCII 2-5
 - character classes 2-11
 - code-set collation order 2-35
 - converting. *See* Code-set conversion.
 - default 2-11
 - destination 2-22, 4-13, 4-45, 4-49, 4-52
 - ISO8859-1 2-11
 - Microsoft 1252 2-11
 - multibyte 2-6, 2-29
 - name of 2-25, 4-13, 4-46, 4-50, 4-53
 - non-state dependent 2-26
 - single-byte 2-5, 2-29, 4-52
 - source 2-22, 4-13, 4-45, 4-49, 4-52
 - state-dependent 2-26
- Code-set collation order 2-35
- Code-set conversion
 - buffer-size estimate 2-24, 2-41
 - conversion table 2-29, 4-52
 - determining if needed 2-22, 4-13
 - fragmented string 2-27
 - INFORMIX-GLS functions for 4-7
 - optimizing 2-29
 - performing 4-45
- single-byte characters 4-52
- state-dependent code sets 2-26
- subdirectory for files 1-13, 1-22
- support for 2-22
- unfragmented string 2-27
- Code-set name registry 2-25
- COLLATE locale category. *See* LC_COLLATE locale-file category.
- Collation
 - code-set collation order 2-35
 - definition of 2-35
 - localized collation order 2-35, 4-152, 4-210
 - types of 2-35
- Comment icons Intro-7
- Compiler
 - built-in scaling 2-5, 2-8, 2-31
- Compiler option
 - DMI_SERVBUILD 1-17
 - I 1-15, 1-17
 - L 1-15
 - lixgls 1-15
- Complementary string 4-158, 4-216
- Compliance
 - with industry standards Intro-15
- Concatenation
 - description of 2-32
 - multibyte-character strings 2-32, 4-147, 4-165
 - wide-character strings 2-32, 4-206, 4-220
- Constant
 - IFX_GL_EINVAL 2-44, 2-45, A-3
 - IFX_GL_NO_LIMIT 2-39
 - IFX_GL_NULL 2-39
- Contact information Intro-16
- Control character 2-12, 2-14, 4-98, 4-124
- Conventions,
 - documentation Intro-5
- Conversion
 - date strings 4-15
 - date/time strings 4-24
 - definition of 3-8
 - INFORMIX-GLS functions for 3-10
 - money strings 4-35
 - number strings 4-40

time strings 4-24
wide-character string to
 multibyte-character
 string 4-233
See also Case conversion; Code-set
 conversion; Formatting.

Converting
 date value to string 4-15
 date/time value to string 4-24
 money value to string 4-35
 number value to string 4-40
 string to date value 4-54
 string to date/time value 4-63
 string to money value 4-74
 string to number value 4-80
 string to time value 4-63
 time value to string 4-24

conv_state_t structure 2-26, 4-47

Copying
 multibyte-character string 2-33,
 4-155, 4-168
 wide-character string 2-33,
 4-213, 4-222

CTYPE locale category. *See*
 LC_CTYPE locale-file category.

Currency data. *See* Monetary data.

Currency symbol 3-5, 4-77

Current processing locale 1-11,
 1-18, 1-20, 2-25, 2-42, 4-89

D

Data formatting 3-3, 4-9

Database locale 1-10, 1-11, 1-18

Database server, client behavior
 on Intro-4

DataBlade client application
 compiling 1-14
 current processing locale 1-11
 initializing INFORMIX-GLS 1-18,
 4-89
 linking INFORMIX-GLS 1-14
 localizing 1-8

DataBlade UDR
 compiling 1-16
 current processing locale 1-11
 initializing INFORMIX-GLS 1-18,
 4-89

linking INFORMIX-GLS 1-16

localizing 1-8

MI_SERVBUILD compiler
 flag 1-17

Date data
 alternative digits for 3-7
 converting 4-15
 era names 3-7
 field width 4-21, 4-60
 format string 4-16, 4-55
 formatting 4-54
 formatting and conversion
 functions 4-10
 formatting directives for 4-17,
 4-19, 4-56, 4-58
 locale category for 3-6
 locale-specific formats 3-6
 month names 3-7
 precision 4-60
 SQL data types for 3-3
 weekday names 3-7
See also Date/time data.

Date string 3-6

Date/time data
 alternative digits for 3-7
 A.M. equivalent 3-7
 converting 4-24
 field width 4-32, 4-70
 format string 4-25, 4-64
 formatting 4-63
 formatting and conversion
 functions 4-10
 formatting directives for 4-27,
 4-30, 4-65, 4-68

locale category for 3-6

locale-specific formats 3-6

precision 4-70

P.M. equivalent 3-7

SQL data types for 3-3
See also Date data.

Date/time string 3-6

DB-Access utility Intro-5

DBCENTURY environment
 variable 4-18, 4-20, 4-29, 4-31

DBDATE environment
 variable 4-16, 4-20, 4-31, 4-55,
 4-59, 4-69

DBMONEY environment
 variable 4-36, 4-75

DBTIME environment
 variable 4-25, 4-31, 4-64, 4-69

DB_LOCALE environment
 variable 1-11

DECIMAL data type
 See Monetary data; Numeric data.

Decimal separator 3-4, 3-5, 4-78

Default locale Intro-4, 1-11,
 3-10, 3-12

Demonstration databases Intro-5

Dependencies, software Intro-4

Digit character
 decimal number 2-12, 2-13, 2-14,
 4-91, 4-93, 4-100, 4-118,
 4-120, 4-126
 hexadecimal number 2-12, 2-14,
 4-115, 4-140

Digit class 2-12, 2-14, 4-91, 4-100,
 4-118, 4-126

-DMI_SERVBUILD compiler
 option 1-17

Documentation notes Intro-14

Documentation notes, program
 item Intro-14

Documentation, related Intro-15

Documentation, types of
 documentation notes Intro-14
 error message files Intro-13
 on-line manuals Intro-12
 printed manuals Intro-13
 related reading Intro-15
 release notes Intro-14

E

End-user format 3-3

English locale
 See Default locale.

Environment variables Intro-6

en_us.8859-1 locale Intro-4

Error handling functions 4-6

Error message files Intro-13, 1-12

Error messages
 internationalizing 1-6
 list of error numbers A-1
 retrieving error numbers 1-19,
 4-142
See also individual function names.

Error number 4-142, A-1
ESQL/C client application
 compiling 1-14
 current processing locale 1-11
 initializing INFORMIX-GLS 1-18,
 4-89
 linking INFORMIX-GLS 1-14
 localizing 1-8
Exception handling 1-19, 4-142

F

Feature icons Intro-8
Features of this product,
 new Intro-5
Field width
 of date string 4-21, 4-60
 of date/time string 4-32, 4-70
 of money string 4-38, 4-78
 of number string 4-43, 4-84
Field width and precision
 of date 4-60
 of datetime 4-70
Fill character 4-77
Find Error utility Intro-13
finderr utility Intro-13
first_frag field 2-26, 4-47
FLOAT data type
 See Numeric data.
Format string 3-9
 date to string 4-55
 date/time to string 4-64
 money to string 4-75
 number to string 4-81
 string to date 4-16
 string to date/time 4-25
 string to money 4-36
 string to number 4-41
 string to time 4-25
 time to string 4-64
Formatting
 date values 4-54
 date/time values 4-63
 definition of 3-8
 INFORMIX-GLS functions
 for 3-11, 4-9
 money values 4-74
 number values 4-80

 time values 4-63
 See also Conversion.
Formatting directive
 date strings 4-56, 4-58
 date values 4-17, 4-19
 date/time strings 4-65, 4-67
 date/time values 4-27, 4-29
 definition of 3-9
 Informix compatible 4-20, 4-31,
 4-59, 4-69
 modified 4-19, 4-29, 4-58, 4-67,
 4-77, 4-83
 money strings 4-76, 4-77
 money values 4-37
 number strings 4-82, 4-83
 number values 4-42
 purpose in conversion 3-10
 purpose in formatting 3-11
 See also Type specifier.
Form-feed character 2-12
 See also Vertical white space.
Fragmented string 2-26, 2-27

G

Global Language Support
 (GLS) Intro-4
GLS locale. *See* Locale.
gls.h header file 1-13, 2-7, 2-9,
4-143, A-1
GL_DATE environment
 variable 4-16, 4-55
GL_DATETIME environment
 variable 4-25, 4-64
gl_mchar_t data type 2-7
gl_wchar_t data type 2-9
Graph class 2-13, 2-14, 4-102, 4-128
Graphical character 2-13, 2-14,
4-102, 4-128
Grouping information 3-4, 3-5
Groupings separator. *See*
 Thousands separator.

H

Header file
 directory location 1-13
 ifxgls.h 1-13, 1-14, 1-17

Hexadecimal-digit character 2-12,
2-14, 4-82, 4-115, 4-140
Horizontal white space
 definition of Intro-10, 2-12
 in format strings 4-16, 4-18, 4-25,
 4-36, 4-41
 representation of Intro-10,
 Intro-11
 testing for 2-13, 4-96, 4-110,
 4-122, 4-136
 trailing 4-160, 4-173, 4-175,
 4-218, 4-224

I

I18n abbreviation 1-4
Icons
 feature Intro-8
 Important Intro-7
 platform Intro-8
 product Intro-8
 Tip Intro-7
 Warning Intro-7
ifxgls.h header file 1-13, 1-14, 1-17,
2-7, 2-9
IFX_GL_CASE_CONV_DST_
 BYTES macro 2-18, 4-197, 4-200
ifx_gl_case_conv_outbufen()
 function 2-16, 2-17, 2-41, 4-11
IFX_GL_CASE_CONV_SRC_
 BYTES macro 2-18, 4-197, 4-200
ifx_gl_convert_datetime()
 function 3-9, 3-10, 4-24
ifx_gl_convert_date() function 3-9,
3-10, 4-15
ifx_gl_convert_money()
 function 3-9, 3-10, 4-35
ifx_gl_convert_number()
 function 3-9, 3-10, 4-40
ifx_gl_conv_needed()
 function 2-22, 2-25, 4-13
ifx_gl_cv_mconv() function 2-25,
2-26, 4-45
ifx_gl_cv_outbufen()
 function 2-23, 2-24, 2-25, 2-41,
 4-49
ifx_gl_cv_sb2sb_table()
 function 2-25, 2-29, 4-52

- IFX_GL_EINVAL constant 2-44, 2-45, A-3
- ifx_gl_format_datetime() function 3-9, 3-11, 4-63
- ifx_gl_format_date() function 3-9, 3-11, 4-54
- ifx_gl_format_money() function 3-9, 3-11, 4-74
- ifx_gl_format_number() function 3-9, 3-11, 4-80
- ifx_gl_getmb() function 1-21, 4-86
- ifx_gl_init() function 1-18, 1-20, 4-89
- ifx_gl_isalnum() function 2-13, 4-91
- ifx_gl_isalpha() function 2-13, 4-93
- ifx_gl_isblank() function 2-13, 4-96
- ifx_gl_iscntrl() function 2-14, 4-98
- ifx_gl_ismdigit() function 2-14, 4-100
- ifx_gl_ismgraph() function 2-14, 4-102
- ifx_gl_ismlower() function 2-13, 4-104
- ifx_gl_ismprint() function 2-14, 4-106
- ifx_gl_ismprint() function 2-14, 4-108
- ifx_gl_ismspace() function 2-13, 4-110
- ifx_gl_ismupper() function 2-13, 4-113
- ifx_gl_ismxdigit() function 2-14, 4-115
- ifx_gl_iswalnum() function 2-13, 4-118
- ifx_gl_iswalpha() function 2-13, 4-120
- ifx_gl_iswblank() function 2-13, 4-122
- ifx_gl_iswcntrl() function 2-14, 4-124
- ifx_gl_iswdigit() function 2-14, 4-126
- ifx_gl_iswgraph() function 2-14, 4-128
- ifx_gl_iswlower() function 2-13, 4-130
- ifx_gl_iswprint() function 2-14, 4-132
- ifx_gl_iswpunct() function 2-14, 4-134
- ifx_gl_iswspace() function 2-13, 4-136
- ifx_gl_iswupper() function 2-13, 4-138
- ifx_gl_iswxdigit() function 2-14, 4-140
- ifx_gl_lc_errno() function 1-19, 1-20, 2-14, 4-142, A-1
See also Error messages.
- ifx_gl_mbllen() function 2-30, 4-145
- ifx_gl_mbscat() function 2-32, 4-147
- ifx_gl_mbschr() function 2-34, 4-150
- ifx_gl_mbscoll() function 2-36, 4-152
- ifx_gl_mbscpy() function 2-33, 4-155
- ifx_gl_mbscspn() function 2-34, 4-157
- ifx_gl_mbslen() function 2-33, 4-160
- ifx_gl_mbsmbs() function 2-34, 4-162
- ifx_gl_mbsncat() function 2-32, 4-165
- ifx_gl_mbsncpy() function 2-33, 4-168
- ifx_gl_mbsnext() function 2-30, 4-145, 4-171
- ifx_gl_mbsnbytes() function 2-33, 4-173
- ifx_gl_mbsnbytes() function 2-33, 4-175
- ifx_gl_mbspbrk() function 2-34, 4-177
- ifx_gl_mbspnext() function 2-30, 4-180
- ifx_gl_mbsrchr() function 2-34, 4-182
- ifx_gl_mbspn() function 2-34, 4-185
- ifx_gl_mbstowcs() function 1-23, 2-10, 4-188
- ifx_gl_mbtowc() function 2-10, 4-191
- ifx_gl_mb_loc_max() function 2-17, 2-41, 4-87, 4-144, 4-237
- IFX_GL_MB_MAX macro 2-17, 2-24, 2-41, 4-87, 4-237
- IFX_GL_NO_LIMIT constant 2-39
- IFX_GL_NULL constant 2-39
- IFX_GL_PROC_CS macro 2-22, 2-25, 4-13, 4-46, 4-50, 4-53
- ifx_gl_putmb() function 1-21, 4-193
- ifx_gl_tomlower() function 2-15, 2-16, 4-196
- ifx_gl_toupper() function 2-15, 2-16, 4-199
- ifx_gl_towlower() function 2-15, 4-202
- ifx_gl_towupper() function 2-15, 4-204
- ifx_gl_wcscat() function 2-32, 4-206
- ifx_gl_wcschr() function 2-34, 4-208
- ifx_gl_wcscoll() function 2-36, 4-210
- ifx_gl_wcsncpy() function 2-33, 4-213
- ifx_gl_wcscspn() function 2-34, 4-215
- ifx_gl_wcslen() function 2-31, 2-33, 4-218
- ifx_gl_wcsncat() function 2-32, 4-220
- ifx_gl_wcsncpy() function 2-33, 4-222
- ifx_gl_wcsnbytes() function 2-33, 4-224
- ifx_gl_wcsnbytes() function 2-34, 4-226
- ifx_gl_wcsrchr() function 2-34, 4-228
- ifx_gl_wcspn() function 2-34, 4-230
- ifx_gl_wcstombs() function 1-23, 2-10, 4-233
- ifx_gl_wcswcs() function 2-34, 4-235

ifx_gl_wctomb() function 2-10, 4-237
 Important paragraphs, icon for Intro-7
 Include file. *See* Header file.
 Industry standards, compliance with Intro-15
 INFORMIXDIR environment
 variable 1-9, 1-12, 1-14, 2-25
 INFORMIXDIR/bin
 directory Intro-5
 INFORMIX-GLS library
 allocating memory for 1-21
 character processing 2-3
 compatibility 1-6
 data formatting 3-3
 error number 4-142
 gl_mchar_t data type 2-7
 gl_wchar_t data type 2-9
 handling exceptions of 1-19, 4-142
 initializing 1-18
 optimization of 1-22
 required directories 1-13
 single-byte characters 2-5
 stream I/O 1-21
 string-processing functions 2-32
 Initialization functions 4-6
 INT8 data type
 See Numeric data.
 INTEGER data type
 See Numeric data.
 International monetary format 3-5, 4-37, 4-76
 Internationalization
 character processing 2-3
 data formatting 3-3
 definition of 1-3
 INFORMIX-GLS library and 1-5
 ISO8859-1 code set Intro-4, 2-11

L

L10n abbreviation 1-8
 last_frag field 2-26, 4-47
 LC_COLLATE locale-file
 category 2-35, 2-36, 4-153, 4-211

LC_CTYPE locale-file category
 alnum class 2-12, 4-92, 4-118
 alpha class 2-11, 4-94, 4-121
 blank class 2-12, 4-97, 4-122
 cntrl class 2-12, 4-99, 4-124
 description of 2-11
 digit class 2-12, 4-101, 4-127
 graph class 2-13, 4-103, 4-128
 lower class 2-11, 4-105, 4-131
 print class 2-13, 4-107, 4-132
 punct class 2-13, 4-109, 4-135
 space class 2-12, 4-111, 4-137
 upper class 2-12, 4-114, 4-139
 use in case conversion 2-15
 use in character
 classification 2-14
 xdigit class 2-12, 4-116, 4-141
 LC_MONETARY locale-file
 category
 currency_symbol
 subcategory 3-5, 4-37, 4-76
 description of 3-5
 int_curr_symbols subcategory 3-5, 4-37, 4-76
 subcategories of 3-5
 use in conversion 4-36, 4-38
 use in formatting 4-75, 4-79
 LC_NUMERIC locale-file category
 description of 3-4
 subcategories of 3-4
 use in conversion 4-41, 4-44
 use in formatting 4-81, 4-84
 LC_TIME locale-file category
 abday subcategory 3-7, 4-17, 4-27, 4-56, 4-65
 abmon subcategory 3-7, 4-17, 4-27, 4-56, 4-65
 alt_digits subcategory 3-7, 4-19, 4-29, 4-58, 4-67
 am_pm subcategory 3-7, 4-28, 4-66
 converting date values 4-21
 converting date/time values 4-33
 day subcategory 3-7, 4-17, 4-27, 4-56, 4-65
 description of 3-6
 d_fmt subcategory 3-7, 4-16, 4-18, 4-28, 4-55, 4-57, 4-67

 d_t_fmt subcategory 3-7, 4-25, 4-27, 4-64, 4-66
 era subcategory 3-7, 4-19, 4-20, 4-30, 4-58, 4-68
 era_d_fmt subcategory 3-7, 4-19, 4-30, 4-58, 4-68
 era_d_t_fmt subcategory 3-7, 4-30, 4-68
 era_t_fmt subcategory 3-7, 4-30, 4-68
 formatting date values 4-61
 formatting date/time values 4-71
 mon subcategory 3-7, 4-17, 4-27, 4-56, 4-65
 subcategories of 3-7
 t_fmt subcategory 3-7, 4-28, 4-67
 t_fmt_ampm subcategory 3-7, 4-28, 4-66
 Length-terminated string 2-38
 Locale Intro-4, 1-11
 character classes 2-11
 choosing 1-9
 client 1-10, 1-11
 code set. *See* Code set.
 current 2-25
 database 1-10, 1-11
 default 1-11
 definition of 1-9
 determining maximum character size 2-41
 effect on collation 4-153, 4-211
 effect on date-string
 conversion 4-21
 effect on date-string
 formatting 4-61
 effect on date/time-string
 conversion 4-33
 effect on date/time-string
 formatting 4-71
 effect on money-string
 conversion 4-38
 effect on money-string
 formatting 4-79
 effect on number-string
 conversion 4-44
 effect on number-string
 formatting 4-84
 initializing 4-89
 LC_COLLATE category 2-35

LC_CTYPE category 2-11
 LC_MONETARY category 3-5
 LC_NUMERIC category 3-4
 LC_TIME category 3-6
 localized collation order 2-35
 maximum number of bytes in characters 4-144
 name of 2-25, 4-13, 4-46, 4-50, 4-53
 server 1-10, 1-11
 server-processing 1-18
 subdirectory for files 1-13, 1-22
 U.S. English. *See* Default locale.
 white space Intro-10
 Localization 1-8
 Localized collation order 2-35, 4-152, 4-210
 Lower class 2-11, 2-13, 4-104, 4-130
 Lowercase character
 converting from uppercase 2-15, 4-196, 4-202
 converting to uppercase 2-15, 4-199, 4-204
 description of 2-11
 testing for 2-13, 4-104, 4-130
 LVARCHAR data type
 See Character; Character string.

M

Macro
 IFX_GL_CASE_CONV_DST_BYTES 2-18, 4-197, 4-200
 IFX_GL_CASE_CONV_SRC_BYTES 2-18, 4-197, 4-200
 IFX_GL_MB_MAX 2-17, 2-24, 2-41
 IFX_GL_PROC_CS 2-22, 2-25, 4-13, 4-46, 4-50, 4-53
 Memory management
 by INFORMIX-GLS library 1-21
 case conversion 2-17, 2-24
 code-set conversion 2-24
 deallocating strings 2-43
 fragmented string and 4-47
 getting a multibyte character 4-87
 INFORMIX-GLS functions for 4-5
 multibyte-character strings and 2-41
 wide character to multibyte character conversion 4-237
 wide-character strings and 2-42
 Message file
 location of 1-12
 requirements 1-12
 See also Error messages.
 Message file for error messages Intro-13
 Microsoft 1252 code set 2-11
 MI_SERVBUILD compiler flag 1-17
 Monetary data
 converting 4-35
 currency symbol 3-5
 decimal separator 3-5
 field width 4-38, 4-78
 format string 4-36, 4-75
 formatting 4-74
 formatting and conversion functions 4-10
 formatting directives for 4-37, 4-76
 grouping information 3-5, 4-38
 international format 3-5, 4-37, 4-76
 locale category for 3-5
 locale-specific formats 3-5
 national format 3-5, 4-37, 4-76
 negative sign 3-5
 positive sign 3-5
 precision 4-78
 SQL data types for 3-3
 thousands separator 3-5, 4-77
 MONETARY locale category. *See* LC_MONETARY locale-file category.
 MONEY data type
 See Monetary data.
 Money string 3-5
 Multibyte character
 alphabetic 2-13, 4-93, 4-104, 4-113
 alphabetic or digit 2-13, 4-91
 blank character 2-13, 4-96
 case conversion 2-16, 2-23, 4-196, 4-199
 classifying 2-13
 control character 2-14, 4-98
 converting from wide character 2-10, 4-237

converting to wide character 2-10, 4-191
 data type for 2-7
 decimal digit 2-14, 4-100
 definition of 2-6
 description of 2-6
 getting 1-21, 4-86
 graphical character 2-14, 4-102
 hexadecimal digit 2-14, 4-115
 horizontal-space character 2-13, 4-96
 inputting 1-21, 4-86
 locating in character string 2-34, 4-150, 4-182
 lowercase character 2-13, 4-104
 maximum size in bytes 2-41, 4-144
 null byte in 2-6
 null-terminated 2-39
 number of bytes in 2-30, 4-145
 outputting 1-21, 4-193
 passing to function 2-6
 printable 2-14, 4-106
 punctuation 2-14, 4-108
 putting 1-21, 4-193
 representation of Intro-9
 splitting 2-43
 stream I/O 1-21
 termination of 2-39
 uppercase character 2-13, 4-113
 vertical-space character 2-13, 4-110
 white-space character 2-13, 4-110
 Multibyte-character string
 buffer-size considerations 2-43
 case conversion 2-19
 comparing 2-36, 4-152
 concatenation 2-32, 4-147, 4-165
 converting to wide-character string 4-188
 copying 2-33, 4-155
 copying substring 4-168
 data type for 2-7
 fragmenting 2-44
 functions for processing 2-32
 keeping consistent 2-43
 length of 2-33, 4-160
 length without trailing space 2-33, 4-173, 4-175

length-terminated 2-38, 2-40, 2-44, 2-45
 locating any character from substring 4-177
 locating first occurrence of character in 2-34, 4-150
 locating last occurrence of character in 2-34, 4-182
 locating specific character in 2-34
 locating substring in 2-34, 4-162
 memory allocation 2-41
 null terminator 2-6
 null-terminated 2-38, 2-40, 2-42, 4-145
 number of bytes in 2-33, 4-173
 number of characters in 2-33, 4-160
 number of characters without trailing space 2-33, 4-175
 number of matching characters 2-34, 4-185
 number of unmatched characters 2-34, 4-157
 pointer to next character 2-30, 4-171
 pointer to previous character 2-30, 4-180
 sorting 2-36
 string termination 2-38
 traversing 2-30, 4-145
 truncating 2-44
 unfragmented 2-27

N

National currency format 3-5, 4-37, 4-76
 NCHAR data type
 See Character; Character string.
 Negative sign 3-4, 3-5
 New features of this product Intro-5
 Newline character 2-12, 4-57, 4-66
 See also Vertical white space.
 Null character 2-6, 2-38
 Null terminator 2-6, 2-42, 4-55, 4-64, 4-75, 4-81
 Null-terminated string 2-39

Number string 3-4
 Numeric data
 base-4 representation 4-43, 4-82
 binary representation 4-42, 4-82
 converting 4-40
 decimal separator 3-4
 definition of 3-4, 3-5
 field width 4-43, 4-84
 floating-point
 representation 4-42, 4-82
 format string 4-41, 4-81
 formatting 4-80
 formatting and conversion functions 4-10
 formatting directives for 4-42, 4-82
 grouping information 3-4
 hexadecimal representation 4-43, 4-82
 locale category for 3-4
 locale-specific formats 3-4
 negative sign 3-4
 octal representation 4-43, 4-82
 positive sign 3-4
 precision 4-84
 SQL data types for 3-3
 thousands separator 3-4
 unsigned decimal
 representation 4-43, 4-82
 NUMERIC locale category. *See* LC_NUMERIC locale-file category.
 Numeric-fill character. *See* Fill character.
 NVARCHAR data type
 See Character; Character string.

O

On-line manuals Intro-12
 Optimizing. *See* Performance.

P

Percent symbol (%) 3-9
 Performance
 advancing pointer 4-145
 case conversion 2-21

code-set conversion 2-29
 improving 1-22
 INFORMIX-GLS library
 optimization 1-22
 wide-character processing 1-23
 Platform icons Intro-8
 Positive sign 3-4, 3-5
 Precision
 of date string 4-60
 of date/time string 4-70
 of money string 4-78
 of number string 4-84
 Print class 2-13, 2-14, 4-106, 4-132
 Printable character 2-13, 2-14, 4-106, 4-132
 Printed manuals Intro-13
 Product icons Intro-8
 Program group
 Documentation notes Intro-14
 Release notes Intro-14
 Punct class 2-13, 2-14, 4-108, 4-134
 Punctuation character 2-13, 2-14, 4-108, 4-134

R

Radix character. *See* Decimal separator.
 Related reading Intro-15
 Release notes Intro-14
 Release notes, program item Intro-14
 Resource file 1-12
 Return values. *See individual function names.*
 rofferr utility Intro-13

S

Server locale 1-10, 1-11, 1-18
 Server-processing locale 1-10, 1-11, 1-18
 SERVER_LOCALE environment variable 1-11
 Single-byte character
 description of 2-5
 forward traversal of 2-30, 4-171
 in code-set conversion 4-52

representation of Intro-9
 reverse traversal of 4-180
 SMALLFLOAT data type
 See Numeric data.
 SMALLINT data type
 See Numeric data.
 Software dependencies Intro-4
 Sort order. *See* Collation.
 Space character. *See* Blank character; White-space character.
 Space class 2-12, 2-13, 4-28, 4-110, 4-136
 Stream input 1-21
 Stream output 1-21
 String operations 2-30
 See also String processing; String traversal.
 String processing
 character searching 2-34
 concatenation 2-32
 INFORMIX-GLS functions for 2-32
 multibyte characters 2-32
 string copying 2-33
 string-length determination 2-33
 wide characters 2-32
 String traversal 2-30
 String. *See* Character string; Multibyte-character string; Wide-character string.
 Structure
 conv_state_t 2-26, 4-47
 gl_wchar_t 2-9
 System requirements
 database Intro-4
 software Intro-4

T

Tab character 2-12, 4-57, 4-66
 See also Horizontal white space.
 Thousands separator 3-4, 3-5, 4-77, 4-78
 Thread-safe application 1-21
 Time data. *See* Date / time data.
 TIME locale category. *See* LC_TIME locale-file category.

Tip icons Intro-7
 Trailing white space
 including in length of multibyte-character string 4-160
 including in length of wide-character string 4-218
 omitting from length of multibyte string 4-173, 4-175
 omitting from length of wide-character string 4-224
 representation of Intro-11
 Type specifier
 definition of 3-9
 for converting date strings 4-17
 for converting date / time strings 4-27
 for converting money strings 4-37
 for converting number strings 4-42
 for formatting date strings 4-56
 for formatting date / time strings 4-65
 for formatting money strings 4-76
 for formatting number strings 4-82
 See also Formatting directive.

U

UDR. *See* DataBlade UDR.
 UNIX environment, default code set 2-11
 Upper class 2-12, 2-13, 4-113, 4-138
 Uppercase character
 converting from lowercase 2-15, 4-199, 4-204
 converting to lowercase 2-15, 4-196, 4-202
 description of 2-12
 testing for 2-13, 4-113, 4-138
 User-defined routine. *See* DataBlade UDR.
 Users, types of Intro-3

V

VARCHAR data type
 See Character; Character string.
 Vertical white space 2-12, 2-13, 4-110, 4-136
 Vertical-tab character 2-12
 See also Vertical white space.

W

Warning icons Intro-7
 White-space character
 horizontal or vertical white space 2-12
 horizontal white space 2-12
 See also Horizontal white space; Vertical white space.
 Wide character
 alphabetic 2-13, 4-120, 4-130, 4-138
 alphabetic or digit 2-13, 4-118
 assigning value to 2-9
 blank character 2-13, 4-122
 case conversion 2-20, 4-202, 4-204
 classifying 2-13
 comparing 2-9
 control character 2-14, 4-124
 converting from multibyte character 2-10, 4-191
 converting to multibyte character 2-10, 4-237
 data type for 2-9
 decimal digit 2-14, 4-126
 definition of 2-8
 description of 2-8
 graphical character 2-14, 4-128
 hexadecimal digit 2-14, 4-140
 horizontal-space character 2-13, 4-122
 locating in character string 2-34, 4-208, 4-228
 lowercase character 2-13, 4-130
 printable 2-14, 4-132
 processing of 1-23
 punctuation 2-14, 4-134
 uppercase character 2-13, 4-138

- vertical-space character 2-13, 4-136
- white-space character 2-13, 4-136
- Wide-character string
 - comparing 2-36, 4-210
 - concatenation 2-32, 4-206, 4-220
 - converting to multibyte-character string 4-233
 - copying 2-33, 4-213
 - copying substring 4-222
 - data type for 2-9
 - functions for processing 2-32
 - length of 2-31, 2-33, 4-218
 - length without trailing space 2-33, 4-224
 - length-terminated 2-38
 - locating any character from substring 4-226
 - locating first occurrence of character in 2-34, 4-208
 - locating last occurrence of character in 2-34, 4-228
 - locating specific character in 2-34
 - locating substring in 2-34, 4-235
 - memory allocation 2-42
 - null-terminated 2-38, 2-43
 - number of characters in 2-31, 2-33, 4-218
 - number of characters without trailing space 2-33, 4-224
 - number of matching characters 2-34, 4-230
 - number of unmatched characters 2-34, 4-215
 - sorting 2-36
 - string termination 2-38
 - traversing 2-31
- Windows environments
 - default code set 2-11

X

- Xdigit class 2-12, 2-14, 4-115, 4-140
- X/Open compliance level Intro-15
- X/Open XPG4 internationalization standards 1-8
- X/Open, differences from 1-8