



UniVerse

NLS Guide

Version 10.3
February, 2009

IBM Corporation
555 Bailey Avenue
San Jose, CA 95141

Licensed Materials – Property of IBM

© Copyright International Business Machines Corporation 2008, 2009. All rights reserved.

AIX, DB2, DB2 Universal Database, Distributed Relational Database Architecture, NUMA-Q, OS/2, OS/390, and OS/400, IBM Informix®, C-ISAM®, Foundation.2000™, IBM Informix® 4GL, IBM Informix® DataBlade® module, Client SDK™, Cloudscape™, Cloudsync™, IBM Informix® Connect, IBM Informix® Driver for JDBC, Dynamic Connect™, IBM Informix® Dynamic Scalable Architecture™ (DSA), IBM Informix® Dynamic Server™, IBM Informix® Enterprise Gateway Manager (Enterprise Gateway Manager), IBM Informix® Extended Parallel Server™, i.Financial Services™, J/Foundation™, MaxConnect™, Object Translator™, Red Brick® Decision Server™, IBM Informix® SE, IBM Informix® SQL, InformiXML™, RedBack®, SystemBuilder™, U2™, UniData®, UniVerse®, wIntegrate® are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

This product includes cryptographic software written by Eric Young (eay@cryptosoft.com).

This product includes software written by Tim Hudson (tjh@cryptosoft.com).

Documentation Team: Claire Gustafson Shelley Thompson, Anne Waite

US GOVERNMENT USERS RESTRICTED RIGHTS

Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Table of Contents

Preface

Organization of This Manual	ix
Related Documentation	xiv

Chapter 1

What is NLS?

NLS Mode	1-3
How NLS Mode Works	1-4
Internal Character Set	1-4
Mapping	1-5
Enabling NLS Mode	1-5
What You Get with NLS	1-6
Maps	1-6
Locales	1-6

Chapter 2

Getting Started

Installing NLS	2-3
Removing NLS	2-4
NLS Shared Memory Segments on UNIX Systems	2-4
Making a Plan	2-5
Setting Configurable Parameters	2-6
Editing the <i>uvconfig</i> File	2-8
Setting Default Maps and Locales	2-10
Setting Locales	2-13
Associating Maps with Devices	2-14
Mapping in the Spool Queue	2-15
Setting File Maps	2-16
Setting Terminal Maps	2-17
Retrieving Terminal Settings	2-18
Setting Maps on Tapes and Other Devices	2-19
Updating Accounts	2-20

Configuring NLS for Client Programs	2-21
Maps for Client Programs	2-21
Locales for Client Programs	2-23
Configuration Checklist	2-25

Chapter 3

Maps

How Maps Work	3-3
Main Maps and Input Maps	3-3
Base Maps	3-3
Creating a Map Description	3-8
Example of a Map Description Record	3-10
Creating a Map Table	3-10
Example of a Map Table Record	3-12
Multibyte NLS Maps and System Delimiters	3-14
Handling Extra Characters	3-16
Defining New Characters	3-16
Maps and Files	3-17
Assigning Maps to New Files	3-17
Modifying File Maps	3-17

Chapter 4

Locales

How Locales Work	4-3
Creating Conventions	4-6
Creating New Locales	4-7
Naming Locales	4-7
Format of Convention Records	4-8
Time Records	4-8
Numeric Records	4-14
Monetary Records	4-15
Ctype Records	4-19
Collate Records	4-23
Collating	4-27
How UniVerse Collates	4-27
Example of Accented Collation	4-28
Example of Cased Collation	4-29
Shared Weights and Blocks	4-29
Contractions and Expansions	4-29
Editing Weight Tables	4-30
Calculating the Overall Weight	4-31
Using Locales	4-33

Retrieving Locale Settings	4-33
Saving and Restoring Locales	4-33
Listing Current Locales	4-34
Changing Current Locales	4-34

Chapter 5 NLS in UniVerse BASIC Programs

How UniVerse BASIC Is Affected	5-4
Using the UVNLS.H Include File	5-4
String Length	5-5
Length of Record IDs	5-6
Display Length in BASIC	5-7
Finding the Display Length of a String	5-7
Formatting a String in Display Positions	5-7
Folding Strings Using Display Positions	5-8
Inputting Using Display Length with INPUTDIP	5-8
Block Size Always in Bytes	5-8
The REMOVE Pointer and Multibyte Character Sets	5-9
Maps in UniVerse BASIC	5-10
Determining a File's Map Name	5-10
Maps for Source Files	5-11
Maps and Devices	5-12
Maps for Auxiliary Devices	5-12
@ Function Codes for Terminal and Auxiliary Maps	5-12
Printing Previously Mapped Data with UPRINT	5-13
Finding the Map Associated with a Print Channel	5-13
Maps for UNIX Pipes	5-13
Unmappable Characters	5-15
Unmappable Characters and WRITE Statements	5-15
Unmappable Characters and READ Statements	5-16
Multinational Characters in UniVerse BASIC	5-18
Editing Multinational Characters	5-18
Generating Characters in External Format	5-20
Generating Characters in Internal Format	5-21
Internal and External String Conversion	5-21
Displaying Records by Character Value	5-24
Exchanging Character Values	5-25
Case Inversion and Deadkey Characters	5-25
BASIC and Locales	5-26
Retrieving Locale Settings	5-26
Saving and Restoring Locales	5-26

	Changing the Current Locale	5-26
Chapter 6	NLS in Client Programs	
	Client Programs	6-4
	Maps	6-4
	Locales	6-5
	System Delimiters and the Null Value	6-5
	UniObjects	6-6
	UniObjects for Java and UniObjects for .NET	6-7
	InterCall Functions	6-8
	UCI Programs	6-9
	Connecting to the Server	6-9
	Requesting an SQLConnect	6-9
	Setting the Map and Locale	6-9
	Values in the UCI Configuration File	6-10
	Interpreting the Map Name	6-11
	Interpreting the Locale Name	6-11
	Using SQLGetInfo	6-11
	BCI Programs	6-12
	Connecting to the Server	6-12
	Requesting an SQLConnect	6-12
	Setting the Locale	6-12
	Interpreting the Locale Name	6-13
	Using SQLGetInfo	6-14
	GCI Subroutines	6-15
	Specifying Maps for GCI Subroutines.	6-15
	Data Types for Multibyte Characters	6-15
Chapter 7	NLS Administration Menus	
	Unicode Menu.	7-3
Appendix A	The NLS Database	
Appendix B	National Convention Hooks	
	General Hook Mechanism	B-2
	Support from UniVerse	B-5
	Memory Management	B-6
	Using Hooks in UniVerse	B-7
	Create a GCI Definition for the Initialization Routine	B-7
	Compile the Hook Library	B-8

Build the Hook Library	B-8
Test the Hooks	B-8
Install the Hook Library	B-9
NLS Hook Interface Definitions	B-10
Hook Functions	B-11

Appendix C NLS Quick Reference

UniVerse Commands	C-2
BASIC Statements and Functions	C-4
Map Tables	C-6
UniVerse Locales	C-8
Unicode Blocks	C-10

Preface

This guide is for users, programmers, and administrators who are familiar with UniVerse and want to use and manage its National Language Support (NLS).

Organization of This Manual

This manual contains the following:

Chapter 1, “[What is NLS?](#),” gives an overview of how NLS works, and describes the NLS features that are included in UniVerse.

Chapter 2, “[Getting Started](#),” describes how to get started using the configurations supplied with UniVerse.

Chapter 3, “[Maps](#),” describes character set maps and how to modify them.

Chapter 4, “[Locales](#),” describes locales and how to modify them.

Chapter 5, “[NLS in UniVerse BASIC Programs](#),” tells how to use NLS in UniVerse BASIC programs.

Chapter 6, “[NLS in Client Programs](#),” tells how to include NLS functionality in client programs.

Chapter 7, “[NLS Administration Menus](#),” describes the structure and content of the NLS Administration menu system.

Appendix A, “[The NLS Database](#),” contains reference information about the files in the NLS database.

Appendix B, “[National Convention Hooks](#),” describes the national convention hooks users can write to implement specific NLS functions and then hook them into UniVerse.

Appendix C, “[NLS Quick Reference](#),” contains reference information about commands, UniVerse BASIC statements, and so on.

The [Glossary](#) defines the NLS terms that are used in this manual.

Documentation Conventions

This manual uses the following conventions:

Convention	Usage
Bold	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates UniVerse commands, keywords, and options; UniVerse BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates UniVerse identifiers such as file names, account names, schema names, and Windows file names and paths.
<i>Italic</i>	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, file names, and paths.
Courier	Courier indicates examples of source code and system output.
Courier Bold	In examples, courier bold indicates characters that the user types or keys the user presses (for example, <Enter>).
[]	Brackets enclose optional items. Do not type the brackets unless indicated.
{ }	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
itemA itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
...	Three periods indicate that more of the same type of item can optionally follow.
->	A right arrow between menu options indicates you should choose each option in sequence. For example, "Choose File -> Exit" means you should choose File from the menu bar, then choose Exit from the File pull-down menu.
␣	Item mark. For example, the item mark (␣) in the following string delimits elements 1 and 2, and elements 3 and 4: 1␣2F3␣4V5

Documentation Conventions

Convention	Usage
F	Field mark. For example, the field mark (F) in the following string delimits elements FLD1 and VAL1: FLD1FVAL1VSUBV1SSUBV2
V	Value mark. For example, the value mark (V) in the following string delimits elements VAL1 and SUBV1: FLD1FVAL1VSUBV1SSUBV2
S	Subvalue mark. For example, the subvalue mark (S) in the following string delimits elements SUBV1 and SUBV2: FLD1FVAL1VSUBV1SSUBV2
T	Text mark. For example, the text mark (T) in the following string delimits elements 4 and 5: 1F2S3V4T5

Documentation Conventions (Continued)

The following are also used:

- Syntax definitions and examples are indented for ease in reading.
- All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.
- Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

UniVerse Documentation

UniVerse documentation includes the following:

UniVerse Installation Guide: Contains instructions for installing UniVerse 10.3.

UniVerse New Features Version 10.3: Describes enhancements and changes made in the UniVerse 10.3 release for all UniVerse products.

UniVerse BASIC: Contains comprehensive information about the UniVerse BASIC language. It is for experienced programmers.

UniVerse BASIC Commands Reference: Provides syntax, descriptions, and examples of all UniVerse BASIC commands and functions.

UniVerse BASIC Extensions: Describes the following extensions to UniVerse BASIC: UniVerse BASIC Socket API, Using CallHTTP, and Using WebSphere MQ with UniVerse.

UniVerse BASIC SQL Client Interface Guide: Describes how to use the BASIC SQL Client Interface (BCI), an interface to UniVerse and non-UniVerse databases from UniVerse BASIC. The BASIC SQL Client Interface uses ODBC-like function calls to execute SQL statements on local or remote database servers such as UniVerse, DB2, SYBASE, or INFORMIX. This book is for experienced SQL programmers.

Administering UniVerse: Describes tasks performed by UniVerse administrators, such as starting up and shutting down the system, system configuration and maintenance, system security, maintaining and transferring UniVerse accounts, maintaining peripherals, backing up and restoring files, and managing file and record locks, and network services. This book includes descriptions of how to use the UniAdmin program on a Windows client and how to use shell commands on UNIX systems to administer UniVerse.

Using UniAdmin: Describes the UniAdmin tool, which enables you to configure UniVerse, configure and manage servers and databases, and monitor UniVerse performance and locks.

UniVerse Transaction Logging and Recovery: Describes the UniVerse transaction logging subsystem, including both transaction and warmstart logging and recovery. This book is for system administrators.

UniVerse System Description: Provides detailed and advanced information about UniVerse features and capabilities for experienced users. This book describes how to use UniVerse commands, work in a UniVerse environment, create a UniVerse database, and maintain UniVerse files.

UniVerse User Reference: Contains reference pages for all UniVerse commands, keywords, and user records, allowing experienced users to refer to syntax details quickly.

Guide to Retrieve: Describes Retrieve, the UniVerse query language that lets users select, sort, process, and display data in UniVerse files. This book is for users who are familiar with UniVerse.

Guide to ProVerb: Describes ProVerb, a UniVerse processor used by application developers to execute prestored procedures called procs. This book describes tasks such as relational data testing, arithmetic processing, and transfers to subroutines. It also includes reference pages for all ProVerb commands.

Guide to the UniVerse Editor: Describes in detail how to use the Editor, allowing users to modify UniVerse files or programs. This book also includes reference pages for all UniVerse Editor commands.

UniVerse NLS Guide: Describes how to use and manage UniVerse's National Language Support (NLS). This book is for users, programmers, and administrators.

UniVerse Security Features: Describes security features in UniVerse, including configuring SSL through UniAdmin, using SSL with the CallHttp and Socket interfaces, using SSL with UniObjects for Java, and automatic data encryption.

UniVerse SQL Administration for DBAs: Describes administrative tasks typically performed by DBAs, such as maintaining database integrity and security, and creating and modifying databases. This book is for database administrators (DBAs) who are familiar with UniVerse.

UniVerse SQL User Guide: Describes how to use SQL functionality in UniVerse applications. This book is for application developers who are familiar with UniVerse.

UniVerse SQL Reference: Contains reference pages for all SQL statements and keywords, allowing experienced SQL users to refer to syntax details quickly. It includes the complete UniVerse SQL grammar in Backus Naur Form (BNF).

Related Documentation

The following documentation is also available:

UniVerse GCI Guide: Describes how to use the General Calling Interface (GCI) to call subroutines written in C, C++, or FORTRAN from BASIC programs. This book is for experienced programmers who are familiar with UniVerse.

UniVerse ODBC Guide: Describes how to install and configure a UniVerse ODBC server on a UniVerse host system. It also describes how to use UniVerse ODBC Config and how to install, configure, and use UniVerse ODBC drivers on client systems. This book is for experienced UniVerse developers who are familiar with SQL and ODBC.

UV/Net II Guide: Describes UV/Net II, the UniVerse transparent database networking facility that lets users access UniVerse files on remote systems. This book is for experienced UniVerse administrators.

UniVerse Guide for Pick Users: Describes UniVerse for new UniVerse users familiar with Pick-based systems.

Moving to UniVerse from PI/open: Describes how to prepare the PI/open environment before converting PI/open applications to run under UniVerse. This book includes step-by-step procedures for converting INFO/BASIC programs, accounts, and files. This book is for experienced PI/open users and does not assume detailed knowledge of UniVerse.

API Documentation

The following books document application programming interfaces (APIs) used for developing client applications that connect to UniVerse and UniData servers.

Administrative Supplement for APIs: Introduces IBM's seven common APIs, and provides important information that developers using any of the common APIs will need. It includes information about the UniRPC, the UCI Config Editor, the *ud_database* file, and device licensing.

UCI Developer's Guide: Describes how to use UCI (Uni Call Interface), an interface to UniVerse and UniData databases from C-based client programs. UCI uses ODBC-like function calls to execute SQL statements on local or remote UniVerse and UniData servers. This book is for experienced SQL programmers.

IBM JDBC Driver for UniData and UniVerse: Describes UniJDBC, an interface to UniData and UniVerse databases from JDBC applications. This book is for experienced programmers and application developers who are familiar with UniData and UniVerse, Java, JDBC, and who want to write JDBC applications that access these databases.

InterCall Developer's Guide: Describes how to use the InterCall API to access data on UniVerse and UniData systems from external programs. This book is for experienced programmers who are familiar with UniVerse or UniData.

UniObjects Developer's Guide: Describes UniObjects, an interface to UniVerse and UniData systems from Visual Basic. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with Visual Basic, and who want to write Visual Basic programs that access these databases.

UniObjects for Java Developer's Guide: Describes UniObjects for Java, an interface to UniVerse and UniData systems from Java. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with Java, and who want to write Java programs that access these databases.

UniObjects for .NET Developer's Guide: Describes UniObjects, an interface to UniVerse and UniData systems from .NET. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with .NET, and who want to write .NET programs that access these databases.

Using UniOLEDB: Describes how to use UniOLEDB, an interface to UniVerse and UniData systems for OLE DB consumers. This book is for experienced programmers and application developers who are familiar with UniVerse or UniData, and with OLE DB, and who want to write OLE DB programs that access these databases.

What is NLS?

NLS Mode	1-3
How NLS Mode Works	1-4
Internal Character Set	1-4
Mapping	1-5
Enabling NLS Mode	1-5
What You Get with NLS	1-6
Maps	1-6
Locales	1-6



This chapter gives an overview of what *NLS* (National Language Support) is, why you need it, how it works, and what you will find when you install NLS.

Note: *This manual uses some terms that may be new to you. When a new term is introduced, it is printed in italic. This means you can find an entry for the term in the [Glossary](#).*

NLS Mode

UniVerse has a special mode that offers National Language Support (NLS). With NLS mode enabled, you can use UniVerse in various languages and countries. You can do the following:

- Input data in many character sets (dependent on your local keyboard)
- Retrieve data and format it using your own conventions or those of another country
- Output data to a screen or printer using the character sets and display conventions of different countries
- Write programs that run in different languages and countries without source changes or recompilation

How NLS Mode Works

NLS mode works by using two types of *character set*:

- The NLS *internal character set*
- *External character sets* that cover the world's different languages

In NLS mode, UniVerse maps between the two character sets when you input data to or output data from a database.

Internal Character Set

In NLS mode, UniVerse stores data using a single, large, internal character set that can represent at least 64,000 characters. Each character in the internal character set has a unique *code point*. This is a number that is by convention represented in hexadecimal format. You can use this number to represent the character in programs. UniVerse easily stores many languages. You can also customize UniVerse to handle less common languages.

About Unicode

The NLS internal character set conforms to the Unicode standard. Unicode defines characters using 16-bit codes in 4-digit hexadecimal format. The Unicode standard gives unique character definitions for many languages, as well as many symbols and special characters.

The Unicode standard forms part of ISO 10646. NLS complies with:

- ISO/IEC 10646-1:1993 Basic Multilingual Plane
- Unicode Version 2.0 (with the exception of Tibetan)

For more information about Unicode, see *The Unicode Standard*, Version 2.0, Addison Wesley, ISBN 0-201-48345-9, or the Unicode Consortium's World Wide Web page at <http://www.unicode.org>.

Mapping

When you need to enter, list, print, or transfer data, NLS maps the data to or from the external character set you want to use. NLS includes map tables for many of the character sets used in the world (see the list in Appendix C, “[NLS Quick Reference](#)”). You can specify mapping for:

- UniVerse files
- Operating system files
- Terminals
- Keyboards and other input devices
- Printers (including auxiliary printers)
- Storage media
- Communications devices

Note: If your files contain only ASCII 7-bit characters, they need not be mapped.



Enabling NLS Mode

After you install NLS, you can enable it for your UniVerse system (as described in Chapter 2, “[Getting Started](#)”). This means that any new accounts you create can use NLS immediately, and you can update existing accounts to recognize NLS.

Warning: *If NLS mode is enabled and you create UniVerse files containing non-ASCII data, a UniVerse system without NLS installed and enabled will not recognize that data.*



You do not need to recompile your existing applications in order to run them on a UniVerse system with NLS enabled, but if you want to use the full capabilities of NLS, you may need to change your applications. For more information about this, see Chapter 5, “[NLS in UniVerse BASIC Programs](#).”

What You Get with NLS

NLS has its own configurable database of UniVerse files in the *nls* subdirectory of the UV account directory. For a description of these files, see Appendix A, “[The NLS Database](#),” and Chapter 7, “[NLS Administration Menus](#).” This database contains:

- Information about the Unicode character set. For more information, see Chapter 3, “[Maps](#).”
- Tables of character set maps. For more information, see Appendix C, “[NLS Quick Reference](#).”
- Tables of locales and national conventions that define how data is formatted for a particular country or area. For more information, see Chapter 4, “[Locales](#).”

When you install UniVerse with NLS enabled, the files in the database are configurable as well. This means you can customize all the *categories* defined in each locale.

Maps

Maps define how UniVerse converts characters in the external character set to the internal character set, and vice versa. The external character set is what the user sees and uses to input data on a keyboard, to print reports, and so on. Appendix C, “[NLS Quick Reference](#),” shows the map tables that are supplied with UniVerse. For more information about specifying the correct map for your system, see [Setting Default Maps and Locales](#) in Chapter 2, “[Getting Started](#).”

Locales

Strictly speaking, a UniVerse NLS *locale* is a set of *national conventions*. A locale is viewed as a separate entity from a character set. You need to consider the language, character set, and conventions for data formatting that one or more groups of people use. You define the character set independently, although for national conventions to work correctly, you must also use the appropriate character sets. For example, Venezuela and Ecuador both use Spanish as their language, but have different data formatting conventions.



Locales do not respect national boundaries. One country may use several locales, for example, Canada uses two and Belgium uses three. Several countries may use one locale, for example, a multinational business could define a worldwide locale to use in all its offices. Appendix C, “[NLS Quick Reference](#),” lists all the locales that are supplied with UniVerse and the territories and languages associated with them.

Note: This manual uses the term territory rather than country to describe an area that uses a locale.

National Conventions

A national convention is a standard set of rules that define data formatting a particular territory uses. NLS supports the following national conventions:

- The format for times and dates
- The format for displaying numbers
- How to display monetary values
- Whether a character is alphabetic, numeric, nonprinting, and so on
- The order in which characters should be sorted (collation)

Time and Date

Most territories have a preferred style for presenting times and dates. For times, this is usually a choice between a 12-hour or 24-hour clock. For dates, there are more variations. Here are some examples of formats used by different locales to express 9.30 p.m. on the first day of April in 1990:

Territory	Time	Date	UniVerse Locale
France	21h30	1.4.90	FR-FRENCH
U.S.	9:30 p.m.	4/1/90	US-ENGLISH
Japan	21:30	90.4.1	JP-JAPANESE

Date and Time Formats

Numeric

This convention defines how numbers are displayed, including:

- The character used as the decimal separator (the radix character)
- The character used as a thousands separator
- Whether leading zeros should be used for numbers 1 through –1

For example, the following numbers can all mean one thousand, depending on the locale you use:

Territory	Number	UniVerse Locale
Ireland	1,000	IE-ENGLISH
Netherlands	1.000	NL-DUTCH
France	1 000	FR-FRENCH

Numeric Conversion Examples

Monetary

This convention defines how monetary values are displayed, including:

- The character used as the decimal separator. This may differ from the decimal separator used in numeric formats.
- The character used as a thousands separator. This may differ from the thousands separator used in numeric formats.
- The local currency symbol for the territory, for example, \$, £, or ¥.
- The string used as the international currency symbol, for example, USD (US Dollars), NOK (Norwegian Kroner), or ITL (Italian Lire).
- The number of decimal places used in local monetary values.
- The number of decimal places used in international monetary values.
- The sign used to indicate positive monetary values.
- The sign used to indicate negative monetary values.
- The relative positions of the currency symbol and any positive or negative signs in monetary values.

Here are examples of monetary formats different locales use:

Currency	Format	UniVerse Locale
U.S. Dollars	\$123.45	US-ENGLISH
French Francs	123,45 F	FR-FRENCH
German Marks	DM123,45	DE-GERMAN
Portuguese Escudos	123\$45 Esc	PT-PORTUGUESE

Monetary Format Examples

Character Type

This convention defines whether a character is alphabetic, numeric, nonprinting, and so on. This convention also defines any casing rules, for example, some letters take an accent in lowercase but not in uppercase.

Collation

This convention defines the order in which characters are collated, that is, sorted. There can be many variations in collation order within a single character set. For example, the character Å follows A in Germany, but follows Z in Sweden. For an explanation of how NLS determines the sort order for an external character set, see [How UniVerse Collates](#) in Chapter 4, “Locales.”

Getting Started

Installing NLS	2-3
Removing NLS	2-4
NLS Shared Memory Segments on UNIX Systems	2-4
Making a Plan	2-5
Setting Configurable Parameters	2-6
Editing the <i>uvconfig</i> File	2-8
Setting Default Maps and Locales	2-10
Setting Locales	2-13
Associating Maps with Devices	2-14
Mapping in the Spool Queue.	2-15
Setting File Maps	2-16
Setting Terminal Maps	2-17
Retrieving Terminal Settings.	2-18
Setting Maps on Tapes and Other Devices	2-19
Updating Accounts	2-20
Configuring NLS for Client Programs	2-21
Maps for Client Programs	2-21
Locales for Client Programs	2-23
Configuration Checklist	2-25

This chapter tells you how to install and configure NLS using the maps and locales supplied with UniVerse. Topics include the following:

- Installing NLS and planning your configuration
- Setting the configurable parameters used by NLS
- How to set up maps for devices and files
- How to set up locales
- How to set the maps for client/server programs

Installing NLS

Before you can use NLS you must install it.

On UNIX Systems

Install NLS from the Package menu of the UniVerse System Administration menu.

On Windows Platforms

Install NLS from the UniVerse installation program at the same time you install UniVerse. When installation is complete, use the NLS Administration menus to configure NLS to suit your system. For more information about installation procedures, see the instructions that come with your CD-ROM. For more information about the NLS Administration menus, see Chapter 7, “[NLS Administration Menus.](#)”

NLS Map for Windows Console

When you install UniVerse on Windows platforms, be sure to set up the NLS map for the console correctly. If the map is not set up correctly, UniVerse commands that are run stand-alone or from UniVerse client/server products, such as UniAdmin, may not display messages correctly.

The default is probably something like PC850 or MS1252; however, this may not be correct for your version of Windows. Choose an appropriate map, which may not be one of the PCxxx or MSxxx maps. For example, for the Korean version of Windows you should use KSC5601.

If you cannot set the map correctly during installation, correct it later as follows:

1. Change directory to the UV account directory, for example, D:\IBM\UV.
2. Decompile the console terminal definition:
bin\uvttdc console > tmpfile
3. Use NOTEPAD to edit *tmpfile*. Change *at80* and *at81* to the name of the required NLS map and save the file.
4. Compile the console terminal definition:
bin\uvtic tmpfile

Removing NLS

You remove NLS from the system as follows:

On UNIX Systems

Choose De-install from the Package menu of the **UniVerse System Administration** menu.

On Windows Platforms

You cannot deinstall NLS from the system. (This is the same as other packages, such as UV/Net.)

NLS Shared Memory Segments on UNIX Systems

On UNIX systems, NLS shared memory segments can be identified by a key of the format *aceexxxx* where *xxxx* is a four-digit number. You can see this by running the UNIX *ipcs* command. Under normal circumstances there will be only one such segment when NLS is turned on.

However, occasionally you may see more than one segment. This happens if a *DBsetup* command is run while users are logged on to UniVerse. *DBsetup* creates a new NLS shared memory segment every time it runs. The old segment disappears as soon as the last user of that segment exits from UniVerse.

Making a Plan

Before you configure NLS you should make a plan. For most sites, configuration is simple. If you intend to switch between unrelated character sets, such as Korean and Greek, your configuration will be more complex. This is the information that you need before you start your configuration:

- The name of the character set maps you want to use for terminals, printers, files, client programs, and GCI subroutines. For a list of map names, see [Map Tables](#) in Appendix C, “[NLS Quick Reference](#).”
- The locales you want to use. For a list of locale names, see [UniVerse Locales](#) in Appendix C, “[NLS Quick Reference](#).”
- How you want programs to behave when they encounter characters that cannot be mapped.

You can note the information you need on the configuration checklist at the end of this chapter.

Setting Configurable Parameters

You can set systemwide defaults for NLS in the *uvconfig* file. The defaults are stored as UniVerse configurable parameters. You can specify:

- Whether NLS mode is on or off
- How UniVerse behaves if a character cannot be mapped during read or write operations
- Default maps for new files
- Default maps for files created outside NLS
- Default maps for terminals and other devices
- Default national conventions

The following table lists the NLS configurable parameters in the *uvconfig* file. The default values in the following table may be different on your system. For a complete list of the configurable parameters, see *Administering UniVerse*.

Parameter	Description
NLSDEFDEVMAP	Specifies the name of the default map to use for device input or output. This map is used for all devices except printers that do not have a map specified in the &DEVICE& file. The ASSIGN MAP command overrides this setting. The default value is ISO8859-1+MARKS.
NLSDEFDIRMAP	Specifies the name of the default map to use for type 1 and type 19 files without assigned maps. This occurs if a type 1 or type 19 file was not created on an NLS system and has not had a map defined for it by the SET.FILE.MAP command. This map applies only to the data in records, not to record IDs. The default value is ISO8859-1+MARKS.
NLSDEFFILEMAP	Specifies the name of the default map to use for hashed files without assigned maps. This occurs if a hashed file was not created on an NLS system and has not had a map defined for it by the SET.FILE.MAP command. The default value is ISO8859-1+MARKS.

NLS Configurable Parameters

Parameter	Description
NLSDEFGCIMAP	Specifies the name of the default map to use for string arguments passed to and from GCI subroutines. This map is used if the GCI subroutine does not explicitly define a map. The default value is ISO8859-1+MARKS.
NLSDEFPTRMAP	Specifies the name of the default map to use for printer output. This map is used if a printer does not have a map defined for it in the &DEVICE& file. The default value is ISO8859-1+MARKS.
NLSDEFSEQMAP	Specifies the name of the default map to use for sequential input or output for files or devices without assigned maps. The SET.SEQ.MAP command overrides this setting. The default value is ISO8859-1+MARKS.
NLSDEFSRVLC	Specifies the name of the default locale to use for passing data to and from client programs. This locale is used if the client program does not specify a server locale. The default value is ISO8859-1+MARKS.
NLSDEFSRVMAP	Specifies the name of the default map to use for passing data to and from client programs. This map is used if the client program does not specify a server map. The default value is ISO8859-1+MARKS.
NLSDEFTERMMAP	Specifies the name of the default map to use for terminal input or output. This map is used if a terminal does not have a map defined for it in its <i>terminfo</i> definition. The SET.TERM.TYPE MAP command overrides this setting. The default value is ISO8859-1+MARKS.
NLSDEFUSRLC	Specifies the default locale. The default value is OFF.
NLSLCMODE	Specifies whether locales are enabled. A value of 1 indicates that locales are enabled; a value of 0 indicates that locales are disabled. The default setting is 0. This parameter has no effect unless NLSMODE is set to 1.
NLSMODE	Turns NLS mode on or off. A value of 1 indicates NLS is enabled, a value of 0 indicates NLS is disabled. If NLS mode is disabled, UniVerse does not check any other NLS parameters.

NLS Configurable Parameters (Continued)

Parameter	Description
NLSNEWDIRMAP	Specifies the name of the map to use for new type 1 and type 19 files created when NLS mode is enabled. This map applies only to the data in records, not to record IDs. The default value is ISO8859-1+MARKS.
NLSNEWFILEMAP	Specifies the name of the map to use for new hashed files created when NLS mode is enabled. A value of NONE (the default value) indicates that data is to be held in the internal UniVerse character set.
NLSOSMAP	Specifies the name of the map to use for file names or record IDs visible to the operating system. This chiefly affects CREATE.FILE and record IDs written to type 1 or type 19 files. The default value is ISO8859-1.
NLSREADELSE	Specifies the action to take if characters cannot be mapped when a record is read by a READ statement. A value of 1 indicates that the READ statement takes the ELSE clause. A value of 0 indicates that unmappable characters are returned as the Unicode replacement character 0xFFFD. The default value is 1.
NLSWRITEELSE	Specifies the action to take if characters cannot be mapped when data is written to a record. A value of 1 indicates that the write aborts or takes the ON ERROR clause (if there is one). A value of 0 indicates that unmappable characters are converted to the file map's unknown character (for example, ?) before writing the record. When this happens, some data may be lost.

NLS Configurable Parameters (Continued)

Editing the *uvconfig* File

Use one of the following methods to set the configurable parameters in the *uvconfig* file:

- Choose Config Editor from the UniAdmin **Main** menu.
- Use the EDIT.CONFIG command.
- Choose Installation -> Edit uvconfig from the NLS Administration menu.

To see a sample of the current parameter settings, use the Config Editor option of UniAdmin, or the UniVerse command CONFIG DATA. For information about UniAdmin, see *Using UniAdmin*. For information about the EDIT.CONFIG and CONFIG commands, see the *UniVerse User Reference*.

Setting Default Maps and Locales

You need to plan what the maps for your files will need once you enable NLS. After enabling NLS, check that terminals, printers, other devices, and existing files have maps set to your requirements. This section describes how to set system-wide defaults for maps and locales. You must be a UniVerse Administrator logged on to the UV account to do this. Later sections describe how to set maps and locales for specific uses.

1. Decide which maps you need. See [Map Tables](#) in Appendix C, “[NLS Quick Reference](#),” for a complete list of UniVerse NLS maps. You can view the maps using the Mappings option of the NLS Administration menu. If you cannot find a suitable map, you can define a new one. For more information about defining maps, see [Creating New Maps](#) in Chapter 3, “[Maps](#).”

If your operating system supports a different character set from any of the maps already chosen, you have to choose and build another map for the operating system.

2. Build the maps by choosing Mappings -> Build from the NLS Administration menu.

3. Set the NLS configurable parameters in the *uvconfig* file as follows:
 - Set NLSMODE to 1. This turns NLS on for the whole system.
 - If you want to use NLS locales, set NLSLCMODE to 1.
 - If you want to specify a default locale for the whole system, set the NLSDEFUSERLC parameter to the name of the locale you want to use. (For a list of locale names, see [UniVerse Locales](#) in Appendix C, “[NLS Quick Reference](#).” For more information about setting locales, see “[Setting Locales](#)” on page 13.)
 - Set the NLSDEFTERMMAP and NLSDEFPTRMAP parameters to the map names you want for your terminals and printers.
 - If you have data in existing files, the file maps should match your terminal map. Set the NLSDEFFILEMAP, NLSDEFDIRMAP, and NLSDEFSEQMAP parameters to this map name.
 - Set the NLSNEWDIRMAP and NLSDEFDEVMAP parameters to match the terminal map.
 - Set the NLSOSMAP parameter to the name of the map for the character set used by the operating system.

Leave the NLSNEWFILEMAP parameter set to NONE. This ensures that new files are created in NLS format. Leave all other NLS parameters set as shipped.

UniVerse checks that all the maps you defined have been built. If not, it builds them for you.

4. Exit UniVerse and enter **bin/uvregen** so that UniVerse recognizes the settings in the configurable parameters when you restart it. Then do one of the following, as appropriate:

On UNIX systems: Shut down and restart UniVerse.

On Windows platforms: Start the UniVerse Resource service. If the UniVerse Resource service is already running, you must stop it first.

5. When you reenter UniVerse, NLS mode is on. You can display the default map name associated with your terminal by entering **TERM**.

Moving NLS Map and Locale Definitions

You can move NLS map and locale definitions from one system to another. Execute the following steps:

1. Create a type 19 file in the UV account of the source system.
2. Copy the definition records from the NLS database files to the type 19 file. For maps, these records come from the NLS.MAP.DESCS and NLS.MAP.TABLES files. For locales, these records come from the NLS.LC.TIME, NLS.LC.NUMERIC, NLS.LC.MONETARY, NLS.LC.CTYPE, and NLS.LC.COLLATE files. You may also need weight table information for your Collate category if you defined specific weight tables.
3. Transfer the type 19 file to the target system.
4. Copy the definitions back into the appropriate NLS files.
5. Use NLS.ADMIN to build the maps and locales.
6. Load the maps and locales into shared memory using the documented method for your operating system (see [Building and Installing Maps](#) in Chapter 3, “Maps,” and [Creating New Locales](#) in Chapter 4, “Locales”).

Setting Locales

UVLANG Environment Variable

To set your initial UniVerse locale, use the UVLANG environment variable. When you start a UniVerse session, UniVerse retrieves the value of the UVLANG variable and checks to see if a locale of the specified name is loaded. If it is, it becomes your current locale.

Direct UniVerse connections (*uvsh*), telnet connections, and BCI connections are all affected by the UVLANG variable.

System Locale

You can set a locale for your whole system with the NLSDEFUSERLC parameter in the *uvconfig* file. This procedure is described in “[Setting Default Maps and Locales](#)” on page 10.

Users can set locales from the UniVerse prompt using the SET.LOCALE command. You can set locales from UniVerse BASIC programs using the SETLOCALE function. You can also set locales from client programs. For more information, see [Locales](#) in Chapter 6, “[NLS in Client Programs](#).”

For more information about the locale database and how to customize locales, see Chapter 4, “[Locales](#).”

Associating Maps with Devices

You can associate a map name with a printer or any other device defined in the &DEVICE& file. To do this, add the map name in field 19 of the device's record in &DEVICE&.

- If a device has a specific map defined in &DEVICE&, all input and output for the device use the map.
- If a device does not have a specific map defined in &DEVICE&, it uses the default specified in the *uvconfig* file. The defaults are specified in the following parameters:
 - NLSDEFPTRMAP, for printers
 - NLSDEFDEVMAP, for other devices

On UNIX Systems

You can specify a map for spooled output in the UniVerse spooler configuration file (the UNIX file */usr/spool/uv/sp.config*). The map should match the mapping specified for the equivalent printer in the &DEVICE& file.

On Windows Platforms

UniVerse supports the following two modes of operation for spooled output (SETPTR mode 1):

- GDI mode, which uses the Windows printer driver
- Raw mode, which sends printed data to the printer without translation

In GDI mode, UniVerse translates printed text into calls to the Windows Graphics Device Interface, constructing a printed image that uses the Windows printer driver.

If NLS is enabled, the printed text is converted to Unicode before being passed to the GDI. The UniVerse administrator must install a font compatible with the characters used in the document.

In raw mode, the UniVerse print processor writes the text directly to the printer.

If NLS is enabled, UniVerse searches the &DEVICE& file for a record with the same name as the Windows printer. If this record defines an NLS map, UniVerse applies the map to the printed text before it is written to the printer. The UniVerse administrator must define a suitable map for the print device being used.

UniVerse examines the default data type that is configured for the printer specified for each new print job in order to select GDI mode or raw mode. You can change the mode using the SETPTR command with the GDI and RAW keywords.

Mapping in the Spool Queue

The spool queue directory holds data in UniVerse internal format. When data reaches the printer, UniVerse maps it to an external character set using the appropriate map for the device. When you spool to a hold file, the spooler stores the data using the map associated with the &HOLD& directory. UniVerse then maps the data again when it reaches the printer.

***Note:** If you use the UNIX command `usp` to spool jobs from outside UniVerse, the print job does not come from an NLS environment and no mapping occurs when the job reaches the printer.*



Setting File Maps

For old files not created under NLS mode, UniVerse uses the default map specified in either the NLSDEFFILEMAP or the NLSDEFDIRMAP parameter in the *uvconfig* file. For new files, UniVerse uses the maps specified in the parameters NLSNEWFILEMAP and NLSNEWDIRMAP. If you want to set a specific map for a file, use the SET.FILE.MAP command. If you want to convert an existing non-NLS file to an NLS file, use the UNICODE.FILE command. For more information about these commands, see the *UniVerse User Reference*.

Setting Terminal Maps

UniVerse specifies the default setting for terminal maps in the *uvconfig* file NLSDEFTERMMAP parameter. (You can also specify terminal maps in the *terminfo* file. See [@ Function Codes for Terminal and Auxiliary Maps](#) in Chapter 5, “NLS in UniVerse BASIC Programs”). If you want to set an explicit terminal map, use the SET.TERM.TYPE command with the following syntax:

```
SET.TERM.TYPE [code] [MAP mapname] [AUXMAP mapname]
```

code specifies the terminal type. It is case-sensitive. If you omit *code*, the current terminal type is used by default.

mapname must be built and loaded into shared memory.

Specify *mapname* as DEFAULT if you want to use the map specified for the corresponding terminal type in the *terminfo* file. But if there is no default map defined in *terminfo*, SET.TERM.TYPE uses the default specified in the *uvconfig* parameter NLSDEFTERMMAP.

If you want to set a map for an auxiliary printer attached to the terminal, use AUXMAP. If you do not specify a map for an auxiliary printer, UniVerse uses the terminal's map.

This example sets a terminal map without changing the terminal type:

```
>SET.TERM.TYPE MAP SHIFT-JIS
```

The next example sets the terminal type to VT220 and sets up an auxiliary printer map. The terminal map is set up from the *terminfo* record or from the parameter NLSDEFTERMMAP.

```
>SET.TERM.TYPE VT220 AUXMAP JIS-EUC
```

For more information about the SET.TERM.TYPE command, see the *UniVerse User Reference*.

Retrieving Terminal Settings

You can use the TERM and GET.TERM.TYPE commands to list the terminal and auxiliary printer map names. For example:

```
>TERM
Terminal Printer
Page width:      80      80
Page depth:      24      66
Page skip :      0
LF delay  :      0
FF delay  :      2
Backspace :      8
Term map   : SHIFT-JIS
AUX map   : JIS-EUC
vt220
>GET.TERM.TYPE
DEC vt200/vt220 8 bit terminal (vt220)
Width : 80
Depth : 24
Map   : SHIFT-JIS
```

Setting Maps on Tapes and Other Devices

You can specify a map name for a tape device using the ASSIGN command. This command overrides any map name given in the &DEVICE& file for *device* until you either unassign the device or specify another map with an ASSIGN command.

The following example assigns the tape device MT0 to tape unit 0 and sets its map so that data is written to the tape in the Korean standard character set KSC5601:

```
>ASSIGN MT0 TO MTU 0 MAP KSC5601
```

Updating Accounts

Once NLS mode is enabled, all users who enter UniVerse have NLS mode on by default. All accounts created after NLS mode is enabled can use NLS commands and functionality.

If you are installing NLS on a system that has previously been running UniVerse without NLS, you must use the `NLS.UPDATE.ACCOUNT` command to update all existing accounts. This command ensures that an account contains all of the correct VOC entries and converts relevant system files for NLS use, for example, `&SAVEDLISTS&`. Run the command in all existing user accounts, including the UV account.

When you run the command in the UV account, it asks you if you want to convert SQL catalog files to NLS format. If you are using SQL, answer **yes**. This enables you to create schema, table, and column names containing multibyte characters. However, UniVerse does not support multibyte SQL identifiers at this release.

Configuring NLS for Client Programs

If you access UniVerse through a client program, you must make sure that the client and the server are working with the same character set and locales. Most client programs access UniVerse through one of the following APIs:

- GCI (General Calling Interface)
- BCI (UniVerse BASIC SQL Client Interface)
- UniVerse ODBC
- UCI (Uni Call Interface)
- InterCall
- UniObjects
- UniObjects for Java
- UniObjects for .NET



***Note:** UniVerse handles all mapping for client programs on the server. This section describes the configuration that the server needs. However, your client program can define the character set and locales it uses, too. For more information about this, see Chapter 6, “NLS in Client Programs.”*

Maps for Client Programs

The NLS.CLIENT.MAPS file defines maps for client programs on the server. You define maps for client programs by choosing Mappings -> Clients -> Create from the NLS Administration menu.

UniVerse prompts for the following information:

- A client type and character set identifier (see below).
- An optional description.
- An NLS map name that corresponds to the character set used on the client. This information enables UniVerse to map the character set used on the client to the NLS maps known to UniVerse. For a list of the map tables supplied with NLS, see Appendix C, “NLS Quick Reference.”

The client type and character set identifier are in the following format:

client.type : char.set.ID

client.type identifies the type of client system and should be one of the following:

Client Type	Description
WIN	For clients using, for example, UniObjects or InterCall programs on Windows platforms.
UNIX	For clients using, for example, BCI and UCI programs on UNIX systems.

Client Types

char.set.ID is a text string that identifies the character set used by the client. On Windows platforms, the identifier is normally an integer, for example, 1252. On UNIX systems, the identifier can be any text. An example of a complete client type and character set identifier is WIN:1252.

Each development environment differs in how you determine which *char.set.ID* to use. For example, you can call something like the `COleControl::AmbientLocaleID` in an OLE application.

If UniVerse cannot find the client type and character set identifier, it uses a default. The default is either WIN:DEFAULT or UNIX:DEFAULT. If these defaults are not available on the system, UniVerse uses the value specified in the *uvconfig* file for the NLSDEFSRVMAP parameter.

Configuring the Code Page on Multibyte Windows Platforms

On Windows platforms, the code page detected by UniVerse client programs may not be the real code page in use. This information is returned by an operating system call and is outside the client's control. The code page information is passed to the server, which looks it up in the NLS.CLIENT.MAPS file, part of the NLS database. If there is no entry in the file, UniVerse selects a default, either from the NLS.CLIENT.MAPS file, if one exists, or from the NLSDEFSRVMAP configurable parameter in the *uvconfig* file. It is possible that the server can select the wrong map for the client.

For example, suppose you are running on the Korean version of Windows. This returns the code page number 1252, though the real code page is 949. The client sends an identifier of WIN:1252 to the server. The server tries to find a record for WIN:1252. If it finds the entry that is shipped with UniVerse, this sets the NLS map to MS1252, which is incorrect. You can do one of three things to resolve the problem:

- Change the WIN:1252 entry to point to the correct NLS map, for example:
 Record id: WIN:1252
 0001: Korean character set
 0002: KSC5601+MARKS
- Delete the WIN:1252 entry and set the WIN:DEFAULT entry to point to the correct NLS map.
- Delete both WIN:1252 and WIN:DEFAULT entries and set the NLSDEFSRVMAP configurable parameter to the correct NLS map.

The first of these options is preferable.

Locales for Client Programs

Locales for client programs are defined in the NLS.CLIENTS.LCS file on the server. You set a locale for a client program by choosing Locales -> Clients -> **Create** from the NLS Administration menu.

The system prompts you to enter the following information:

- A client type and locale identifier (see below).
- An optional description.
- The name of the locale to use for the client program. This must be one of the UniVerse locale names in Appendix C, “[NLS Quick Reference](#).”

The client type and locale identifier are in the following format:

client.type : locale.ID

client.type identifies the type of client system and should be one of the following:

Client Type	Description
WIN	For clients using, for example, UniObjects or InterCall programs on Windows platforms.
UNIX	For clients using, for example, BCI and UCI programs on UNIX systems.

Client Types

locale.ID is a text string that identifies the locale used by the client. On Windows platforms, the identifier is a hexadecimal number, for example, 0409. An example of a complete client type and locale identifier is WIN:0409. On UNIX systems the identifier can be any text string.

Configuration Checklist

Use this checklist when you configure NLS to prepare your files, programs, and system where required.

- Convert old files with UNICODE.FILE.
- Check client programs that use the following APIs:
 - GCI (General Calling Interface)
 - BCI (UniVerse BASIC SQL Client Interface)
 - UniVerse ODBC
 - UCI (Uni Call Interface)
 - InterCall
 - UniObjects
 - UniObjects for Java
 - UniObjects for .NET
- Modify UniVerse BASIC programs to account for the display width of characters.
- Modify UniVerse BASIC programs to change CHAR to UNICHAR to allow the full range of Unicode characters. Use the special characters @FM, @SM, @VM, and so forth for the UniVerse system delimiters.
- If you use transaction logging, you need to set the following configurable parameters in the *uvconfig* file to these values:
 - LOGBLSZ to 2048, or at least to 1024
 - LOGBLNUM to 32, or at least to 16
- Use the BYTE function in your UniVerse BASIC programs if you need byte rather than character operations.
- Rebuild secondary indexes if you converted data files to NLS.

- Convert type 25 and distributed files by creating new NLS files and copying the data into them. The UNICODE.FILE command cannot convert them.

Description and Possible Value	Parameter	Your Value
NLS Mode		
1 = on; 0 = off.	NLSMODE	
Program Behavior for Unmappable Characters		
When reading a file, 1 = use ELSE clause, and 0 = use the Unicode replacement character.	NLSREADELSE	
When writing to a file, 1 = the write fails, and 0 = use the file map's unknown character.	NLSWRITEELSE	
Map for New Files Created with NLS Mode On		
Map name for hashed files.	NLSNEWFILEMAP	
Map name for type 1 and 19 files.	NLSNEWDIRMAP	
Map name for record IDs in type 1 or type 19 files.	NLSOSMAP	
Default Maps for Existing Files Created with NLS Mode Off		
Map name for hashed files.	NLSDEFFILEMAP	
Map name for type 1 and type 19 files.	NLSDEFDIRMAP	
Default Maps for Devices		
Map for devices (except printers).	NLSDEFDEVMAP	
Map for printers.	NLSDEFPTRMAP	

Configurable Parameter Settings in the *uvconfig* File

Description and Possible Value	Parameter	Your Value
Map for terminals.	NLSDEFTERMMAP	
Default Map for Sequential I/O		
Map for sequential I/O for a file or device.	NLSDEFSEQMAP	
Default Maps for GCI Subroutines and Client Programs		
Map for GCI string arguments.	NLSDEFGCIMAP	
Map for client program data.	NLSDEFSRVMAP	
Locale Settings		
Locale mode, 1 = on and 0 = off.	NLSLCMODE	
The default locale for the system.	NLSDEFUSERLC	
Locale for client program data.	NLSDEFSRVLC	

Configurable Parameter Settings in the *uvconfig* File (Continued)

Maps

How Maps Work	3-3
Main Maps and Input Maps	3-3
Base Maps	3-3
Creating a Map Description	3-8
Example of a Map Description Record	3-10
Creating a Map Table	3-10
Example of a Map Table Record	3-12
Multibyte NLS Maps and System Delimiters	3-14
Handling Extra Characters	3-16
Defining New Characters.	3-16
Maps and Files	3-17
Assigning Maps to New Files	3-17
Modifying File Maps	3-17

This chapter provides more detailed information about the maps supplied with UniVerse. The topics covered include:

- How UniVerse maps work
- Map types
- How to create, build, and install maps
- Extending a character set to cover extra characters

How Maps Work

UniVerse provides a set of standard map descriptions and tables. Maps are stored in the following two files in the NLS database:

- NLS.MAP.DESCS holds information about maps, such as whether they are singlebyte or doublebyte, and what replacement character should be used for characters that cannot be mapped.
- NLS.MAP.TABLES holds the character mappings themselves. Each code point in the external character set is mapped to a code point in the UniVerse internal character set. Each map table supplied with UniVerse has an entry in this file.

Before you can use a map in a program or a command, you must compile it and load it into shared memory. See [“Building and Installing Maps”](#) on page 13.

Any map name you supply to a program or command must be the ID of a record in the NLS.MAP.DESCS file. Each map record in the file contains a pointer to a *main map table* and optionally to an *input map table* in the NLS.MAP.TABLES file.

Main Maps and Input Maps

Main maps define the input and output mapping for a character set. The mapping is two-way. External byte sequences map to internal values on input, and back to the same external byte sequences on output. For a list of the map tables supplied with UniVerse, see Appendix C, [“NLS Quick Reference.”](#)

Input map tables, also known as *deadkey tables*, are one-way. They define byte sequences that map from external to internal values only. You use them to enter characters that a system can display on the screen but that are not on the keyboard.

Base Maps

A map can be based on another map. When it is, the record in the NLS.MAP.DESCS file also contains a pointer to the *base map*. This map can be based on yet another map. To understand the complete map you must follow the chain of base maps. For more information about the construction of a map, choose Mappings -> Descriptions -> Xref and Mappings -> Tables ->h Xref from the NLS Administration menu.

For example, the map C0-CONTROLS is a singlebyte character set map using the C0-CONTROLS table. It maps the set of 7-bit control characters. The italic comments are not part of the record but are added here for clarity.

```
NLS.MAP.DESCS C0-CONTROLS
  0001 Standard ISO2022 C0 control set, chars 00-1F+7F
  0002                                     - Name of base map
  0003 SBCS
  0004 C0-CONTROLS                         - Name of map table
NLS.MAP.TABLES C0-CONTROLS
  0001 * FIRST 32 CONTROL CHARACTERS (IDENTITY MAP) + DEL
  0002 00-1F 0000
  0003 7F      007F
```

In general, you can construct larger maps from existing maps by adding another table. For example, the map ASCII, which maps all of the 7-bit characters, is constructed by adding the table ASCII to the map C0-CONTROLS:

```
NLS.MAP.DESCS ASCII
  0001 #Standard ASCII 7-bit set
  0002 C0-CONTROLS                         - Name of base map
  0003 SBCS
  0004 ASCII                               - Name of map table
NLS.MAP.TABLES ASCII
  0001 * 7-BIT ASCII, identity mapping to 1st 127 chars
  0002 * (not including control characters - see C0-CONTROLS)
  0003 20-7F 0020
```

Similarly the map C1-CONTROLS, which contains all 8-bit and 7-bit control characters, is constructed by adding the table C1-CONTROLS to the map C0-CONTROLS:

```
NLS.MAP.DESCS C1-CONTROLS
  0001 Standard 8-bit ISO control set, 80-9F
  0002 C0-CONTROLS                         - Name of base table
  0003 SBCS
  0004 C1-CONTROLS                         - Name of map table
NLS.MAP.TABLES C1-CONTROLS
  0001 * ISO 8-BIT 32 CONTROL CHARACTERS (IDENTITY MAP)
  0002 80-9F 0080
```

You can further modify this map as required. The map ASCII+C1 is constructed by adding the table ASCII to the map C1-CONTROLS, and the map ISO8859-1 by adding the table ISO8859-1 to the map ASCII+C1.

Creating a New Map

Complete the following steps to create new maps:

1. Find an existing map that most closely matches the required map.
2. Identify the characters that need to be mapped differently in the new map.
3. Create a new table in NLS.MAP.TABLES that contains only these new mappings.
4. Create the new map in NLS.MAP.DESCS by basing it on the existing map and adding the new table.

The following example creates a map called MY.ASCII. This map is identical to the existing ASCII map, except the input character 0x23 is mapped to the UK pound sign (pound) instead of the number symbol (hash).

```
NLS.MAP.DESCS MY.ASCII
0001 * Modified ASCII with UK pound
0002 ASCII
0003 SBCS
0004 MY.POUND
NLS.MAP.TABLES MY.POUND
0001 * Map input 0x23 to Unicode 00A3
0002 23 00A3
```

Map Naming Conventions

Map names must contain only characters in the ASCII-7 character set. The following map names are reserved and have special meanings:

Map Name	Description
AUX	The map associated with the auxiliary printer.
CRT	The map associated with the current terminal.
DEFAULT	The default map.
LPTR	The map associated with print channel 0.
NONE	No mapping. UniVerse uses the internal character set.
UNICODE	The map from or to the UniVerse internal set and Unicode 16-bit fixed width external set.
UTF8	The map from or to the UniVerse internal set and UTF8 as described in ISO 10646. This involves mapping the UniVerse system delimiters to the Private Use Area of Unicode.

Reserved Map Names

Avoid defining a map that uses any of the following prefixes or suffixes that are associated with existing groups of maps:

Map Name	Description
ASCII...	Underlies most other code pages and defines the characters 0000 through 007F.
BIG5...	The de facto standard Chinese double-byte character set.
EBCDIC...	IBM EBCDIC encodings.
GB...	Chinese GB standards (for example, GB2312-80).
ISO8859- <i>nn</i>	ISO 8859 series of single-byte character set standards.
KSC...	Korean DBCS national standards (for example, KSC5601).

Map Name Reserved Prefixes and Suffixes

Map Name	Description
...JIS and JIS...	Japanese DBCS national standards (for example, SHIFT-JIS and JIS-EUC).
MNEMONICS	A large set of deadkey sequences for entering Unicode characters using the form <xx>. For example, <Ye> enters the Yen symbol.
MAC...	Apple Macintosh code pages (singlebyte character set).
MS <i>nnnn</i>	Microsoft Windows code pages. <i>nnnn</i> is four decimal digits.
PC <i>nnn</i>	IBM PC code pages. <i>nnn</i> is usually three decimal digits.

Map Name Reserved Prefixes and Suffixes (Continued)

Creating New Maps

You can create or edit map records by choosing the Mappings option from the NLS Administration menu. Choose Tables for a map table or Descriptions for a map description. You can then choose one of the following options:

Option	Description
List	Lists all the tables or descriptions.
Create	Creates a new record in the NLS database.
Edit	Edits a record in the NLS database.
Delete	Deletes a record in the NLS database.
Xref	Prints cross-reference information on a record.

Mappings Options

Creating a Map Description

When you create a map description, a new record is added to the NLS.MAP.DESCS file. You are prompted to enter values for the fields in the new record. The following table shows the fields in the file:

Field	Name	Description
0	Map ID	The name used to specify the map in commands and programs.
1	Map Description	A description of the map.
2	Base Map ID	The name of a map to base this one on. This value must be the record ID of another record in the NLS.MAP.DESCS file.
3	Map type	The value of this field must be either SBCS for a singlebyte character set, or DBCS for a doublebyte or multibyte character set. The default value is SBCS.

NLS.MAP.DESCS File Attributes

Field	Name	Description
4	Table ID	The record ID of the map table in the NLS.MAP.TABLES file to which this map description refers. You do not need to specify a value if the map table has the same ID as the map description.
5	Display length	The display length of all characters in the mapping table specified in field 4. Most double-byte character sets have some characters that print as two display positions on a screen (for example, Hangul characters or CJK ideographs). However, the same map will usually require that ASCII characters are printed as one display position. This field does not pick up a value from any base map description. The default value is 1.
6	Unknown char seq.	This field specifies the character sequence to substitute for unknown characters that do not form part of the character set. The value, which is a byte sequence in the external character set, should be a hexadecimal number from one to four bytes. The default value is 3F, the ASCII question mark character. The default is used if neither this map nor any underlying base map has a value in this field.
7	Compose seq.	This field contains the character sequence to compose hexadecimal Unicode values from one to four bytes. If UniVerse detects the sequence on input, the next four bytes entered are checked to see if they are hexadecimal values. If so, the Unicode character with that value is entered directly. If neither this map nor any base map has a value in this field, you cannot input Unicode characters by this means. A value of NONE overrides a compose sequence set by an underlying map.

NLS.MAP.DESCS File Attributes (Continued)

Field	Name	Description
8	Input Table ID	The name of a map table in NLS.MAP.TABLES to be used for inputting deadkey sequences.
9	Prefix string	A string in hexadecimal numbers to be prefixed to all external character mappings in the table referenced by field 4. Used mainly for mapping Japanese character sets.
10	Offset value	A value in hexadecimal numbers to be added to each external mapping in the table referenced by field 4. If prefixed by a minus sign, the value is subtracted. Used mainly for mapping Japanese character sets.

NLS.MAP.DESCS File Attributes (Continued)

Example of a Map Description Record

This example shows the map description record for a custom map for a Korean character set. The italic comments are not part of the actual record, but are added here for clarity.

```

0001: #KOREAN: EUC as described by KSC standard + local changes
0002: KSC5601 - map description record this is based on
0003: DBCS - this map is multibyte
0004: KSC-CHANGES - main table added to KSC5601
0005: 2 - all its characters are double-width
0006: A3BF - FULLWIDTH QUESTION MARK in KSC5601 code
0007: 5C5C - compose sequence is two backslashes \\  

0008: MNEMONICS - name of the input table for deadkeys
0009: - not used
0010: - not used

```

Creating a Map Table

When you create a map table, a new record is added to the NLS.MAP.TABLES file. This is a type 19 file. Records in the file contain comments, and mappings between the external character set and a Unicode code point. The mappings each occupy a single line and can be in any order.

- Blank lines and lines starting with # or * are treated as comments.

- Mapping lines must contain only two values:
 - The first value represents a byte sequence of up to eight bytes in the external character set.
 - The second value is its corresponding Unicode character value.
- Each value must be in hexadecimal notation and can be preceded by the characters 0x.
- The two values must be separated by at least one space or tab.
- A comment must follow the second value and be separated from it by at least one space or tab.
- The first value can be the start and end value of a range, separated by a hyphen (-). The second value should be a single Unicode value corresponding to the start of the range.
- The second value can be one of the following special strings:

String	Value	Use
@IM	xFF	Item mark
@FM	xFE	Field mark
@VM	xFD	Value mark
@SM	xFC	Subvalue mark
@TM	xFB	Text mark
@6M	xFA	The mark below text mark
@7M	xF9	The mark two below text mark
@8M	xF8	The mark three below text mark
@SQL.NULL	x80	Internal representation of the null value

Special Strings

Example of a Map Table Record

Here is an example of part of a map table record:

```
# Part of the Latin-3 character set ISO8859/3. A contrived
example.
# The next line maps a range of bytes to the Unicode values
# 0080 through 00A0.
82-A0    0080
# The next 3 lines map the bytes A1, A2, and A6.
A1       0126    LATIN CAPITAL LETTER H WITH STROKE
A2       02D8    BREVE
A6       0124    LATIN CAPITAL LETTER H WITH CIRCUMFLEX
# The next 2 lines map control characters to SQL null and field
mark.
80       @SQL.NULL
81       @FM
# The next line uses the explicit hexadecimal form of numbers, and
shows
# how a 2-byte sequence is mapped to a Unicode character:
xA7A7    x4E0
```

Building and Installing Maps

To build a map, choose Mappings -> Build from the NLS Administration menu.

UniVerse prompts you to enter the name of a map description record. It also prompts if you want a detailed report of the build to be written to a record called *mapname* in the NLS.MAP.LISTING file. If you choose this option, when the map is built you are prompted to view it.

If there is a warning or error message, you must fix the problem before the map can be built. You must edit either the map description or the map table records referenced by the map description named in *mapname*.

The report in the NLS.MAP.LISTING file:

- Lists all mapping rules in the order of the external byte sequence
- Adds descriptions of the Unicode characters taken from the NLS.CS.DESCS file

Note: The report can be thousands of lines long for large double-byte character set maps.



Multibyte NLS Maps and System Delimiters

NLS provides maps for a number of multibyte character sets such as Japanese, Chinese, and Korean. On their own these maps do not allow the UniVerse system delimiters to be used (which is also true of the singlebyte maps). However, unlike the singlebyte maps, where it is possible to use the internal values of the system delimiters in the external character set, this is not possible with the multibyte maps because the system delimiters can be misinterpreted as lead bytes of multibyte characters. For this reason, NLS provides versions of all the multibyte maps both with and without the UniVerse system delimiters. The maps provided are as follows:

Without System Delimiters	With System Delimiters
BIG5	BIG5+MARKS
GB2312	GB2312+MARKS
JIS-EUC+	JIS-EUC++MARKS
JIS-EUC	JIS-EUC+MARKS
JIS-EUC2+	JIS-EUC2++MARKS
JIS-EUC2	JIS-EUC2+MARKS
KSC5601	KSC5601+MARKS
PRIME-SHIFT-JIS	PRIME-SHIFT-JIS+MARKS
SHIFT-JIS	SHIFT-JIS+MARKS
TAU-SHIFT-JIS	TAU-SHIFT-JIS+MARKS

Provided Maps

The UniVerse system delimiters are mapped into the following values for each character set:

Value (in hex)	System Delimiters
1A	Text mark
1C	Subvalue mark
1D	Value mark
1E	Field mark
1F	Item mark

System Delimiter Mappings

In addition, the null value is mapped to the hexadecimal value 19.

Handling Extra Characters

The character set mapping you want to use may not cover all the characters you need. First check to see if the characters are already defined in a different area of Unicode.

For example, the Hangeul language character set KSC5601-1987 supports only the 4500 Hangeul characters in daily use in Korea; many rarely used or historical characters are omitted. However, Unicode supports over 11,000 Hangeul characters. If you have a Korean system that supports more Hangeul than is available in KSC5601-1987, the characters you need are probably already available in Unicode.

The same applies to Japanese Kanji and Korean Hanja, where the character you want may be available as part of the unified Chinese character set.

Defining New Characters

If Unicode does not define the character you need, you can create a character definition. Unicode has a Private Use Area with values xE000 through xF8FF. This area has room for an additional 6400 characters. You can choose a Unicode value in that area and map your character to it.

The Unicode standard reserves the area from F8FF downward for corporate use, and from E000 upward for individual users' use. UniVerse uses the values F8F7 through F8FF for the UniVerse system delimiters.

Warning: *Take care when transferring data between sites. Both sites must agree on the use of positions E000 upward in the Private Use Area, otherwise you lose data integrity.*



Maps and Files

In NLS mode, each UniVerse file has an associated map that defines the external character set for the file. The maps are stored as follows:

- For type 1 and type 19 files, the map is stored as a file in the O/S directory.
- For all other UniVerse file types, the map name is stored in the file header.

Any files created with NLS mode turned off use the default maps defined by the configurable parameters in the *uvconfig* file.

Assigning Maps to New Files

When you create a new UniVerse file, the `CREATE.FILE` command assigns a default map name to the file. The default map name is defined in the *uvconfig* file as follows:

- The `NLSNEWFILEMAP` parameter defines the value for hashed files.
- The `NLSNEWDIRMAP` parameter defines the values for type 1 and type 19 files.

Modifying File Maps

If you use a UniVerse BASIC program to open and read a file, you must ensure that the file map is the one that your program expects. You can use a call to the `FILEINFO` function to determine the map name. A file's map name is also included in reports generated by the `ANALYZE.FILE`, `FILE.STAT`, and `GET.FILE.MAP` commands.

The `GET.FILE.MAP` command retrieves the name of the map associated with a file. If there is no map name associated with the file, the command gives the name of the default map to be used.

The `LIST.MAPS` command lists maps that are built and installed. The report includes the name and description for each map.

You need to ensure that the map associated with the file you are working with is the one that you want. Use the `SET.FILE.MAP` command when you need to set or modify the file map.

The SET.SEQ.MAP command specifies the map for you to use with UniVerse BASIC sequential I/O statements if you cannot find an explicit map in the sequential file that you open.

Use the UNICODE.FILE command to convert a mapped file to an unmapped file, or vice versa, without making a copy of the file. The conversion process first checks that all record IDs and data can be read from the file using the correct map. If record IDs and data cannot be retrieved using the input map, the command fails. If some characters cannot be converted using the output map, the records are not written.

For full details about these file map commands, see the *UniVerse User Reference*.

Locales

How Locales Work	4-3
Creating Conventions	4-6
Creating New Locales	4-7
Naming Locales	4-7
Format of Convention Records	4-8
Time Records	4-8
Numeric Records	4-14
Monetary Records	4-15
Ctype Records	4-19
Collate Records	4-23
Collating	4-27
How UniVerse Collates	4-27
Example of Accented Collation	4-28
Example of Cased Collation	4-29
Shared Weights and Blocks	4-29
Contractions and Expansions	4-29
Editing Weight Tables	4-30
Calculating the Overall Weight	4-31
Using Locales	4-33
Retrieving Locale Settings	4-33
Saving and Restoring Locales	4-33
Listing Current Locales	4-34
Changing Current Locales	4-34

This chapter provides more information about how locales work, and how to modify the locales and conventions supplied with UniVerse. The topics covered include:

- Creating locales and conventions
- The format of convention records
- How UniVerse collates

How Locales Work

It is important to distinguish between a locale, a category, and a convention.

- A locale comprises a set of categories.
- A category comprises a set of conventions.
- A convention is a rule describing how data values are input or displayed.

In NLS each locale comprises five categories:

- Time
- Numeric
- Monetary
- Ctype
- Collate

Each category comprises various conventions specific to the type of data in each category.

For example, conventions in the Time category include the names of the days of the week, the strings used to indicate AM or PM, the character that separates the hours, minutes, and seconds, and so forth. This information is stored in files in the NLS database.

The following example shows the record for the US-ENGLISH locale:

```
Locale name..... USA
Description..... Country=USA, Language=English
Time/Date..... US-ENGLISH
Numeric..... DEFAULT
Monetary..... USA
Ctype..... DEFAULT
Collate..... DEFAULT
.
.
.
```

Each of the five categories has its own UniVerse file that stores the definitions for these categories. The conventions are grouped together and identified by a name which is the record ID of an item in the appropriate category file.

For example, the US-ENGLISH conventions for Time /Date are defined by a record ID of that name in the NLS.LC.TIME file.

The NLS.LC.ALL file acts as an index for the locales. It contains a record for each locale, such as US-ENGLISH, with fields for each category.

Each field contains a pointer to a record in another file, which is the relevant category file. The Time field has a pointer to a record in the NLS.LC.TIME file, the Numeric field has a pointer to a record in the NLS.LC.NUMERIC file, and so on.

Each category field...	Points to a record in the corresponding file...	The US-ENGLISH locale record contains these corresponding values...
Time	NLS.LC.TIME	USA
Numeric	NLS.LC.NUMERIC	DEFAULT
Monetary	NLS.LC.MONETARY	USA
Ctype	NLS.LC.CTYPE	DEFAULT
Collate	NLS.LC.COLLATE	DEFAULT

NLS Pointers

This means that a locale can be built from existing conventions without duplication. Different locales can share conventions, and one convention can be based on another.

For example, Canada uses the locales CA-FRENCH and CA-ENGLISH. The two locales are not completely different; they share the same Monetary convention. The records in the NLS.LC.ALL file for the CA-FRENCH and CA-ENGLISH locales look like this:

```
Locale name..... CA-FRENCH
Description..... Country=Canada, Language=French
Time/Date..... CA-FRENCH
Numeric..... CA-FRENCH
Monetary..... CANADA
Ctype..... DEFAULT
Collate..... DEFAULT+ACCENT+CASE
.
.
.
Locale name..... CA-ENGLISH
Description..... Country=Canada, Language=English
Time/Date..... CA-ENGLISH
Numeric..... CA-ENGLISH
Monetary..... CANADA
Ctype..... DEFAULT
Collate..... DEFAULT
.
.
.
```

Notice that for both locales the Monetary field points to a record in the NLS.LC.MONETARY file called CANADA. The other fields contain the appropriate value for the language concerned.

You examine the conventions defined for a locale using the NLS Administration menu. Enter the command **NLS . ADMIN** in the UV account, choose **Locales -> View**. When prompted for a locale ID, enter one of the IDs shown in Appendix C, “NLS Quick Reference.”



Note: You must be logged on as a UniVerse Administrator to use NLS.ADMIN. For more information about *NLS Administration menus*, see Chapter 7, “NLS Administration Menus.”

Creating Conventions

The conventions supplied with UniVerse conform to international standards. For major languages you should not need to create completely new conventions. To modify a convention, you create a new convention based on an existing convention. An outline of the procedure is as follows:

1. Plan your new convention. Study the format of the convention records in each category and decide which fields you need to modify. See [“Format of Convention Records”](#) on page 8.
2. From the NLS Administration menu, choose Categories. Then choose Time, Numeric, Monetary, Ctype, or Collate.
3. Using the View option, find a convention that looks like what you need. If you want to create a Collate convention, you may also need to choose a suitable weight table. This is explained in [“Collating”](#) on page 28.
4. Choose the Create option to create the new convention.
5. Choose Edit to change the convention to suit your needs. UniVerse prompts you to edit and save the record using ReVise.

Creating New Locales

To make a new locale from existing conventions:

1. From the NLS Administration menu, choose Locales -> Create. UniVerse prompts you to enter a name for the new locale and the name of an existing locale to base it on.
2. UniVerse then prompts to make any changes to the record using ReVise.
3. Choose Build to build the new locale.

Naming Locales

Locale names can be any string that is a valid UniVerse record ID. You must not use any string that is the same as a VOC record ID. The locales shipped with UniVerse have names that use only ASCII-7 characters, but you can rename them using different character sets, as appropriate.

Format of Convention Records

The following sections describe the fields in convention records in the five categories:

- Time
- Numeric
- Monetary
- Ctype
- Collate

Time Records

Convention records in the Time category are stored in the NLS.LC.TIME file. The following table shows each field number, its display name, and a description for time and date information:

Field	Name	Description
0	Category Name	The name of the convention.
1	Description	A description of the convention. It usually includes the territory that the convention applies to and the language it is used with.
2	Based on	The name of another convention record in the NLS.LC.TIME file on which this convention is based.
3	TIMEDATE format	A format for combined time and date used by the UniVerse BASIC TIMEDATE function and the TIME command. The value should consist of an MT or TI time conversion code, and a D or DI date conversion code. The two codes can be in any order. They should be separated by a tab character, or a text or subvalue mark.

NLS.LC.TIME File Attributes

Field	Name	Description
4	Full DATE format	The full combined date and time format used by the TIME command. The value should consist of an MT or TI time conversion code, and a D or DI date conversion code. The two codes can be in any order. They should be separated by a tab character, or a text or subvalue mark.
5	Date 'D' format	The default date format for the D conversion code. The value should be any D or DI conversion code.
6	Date 'DI' format	The default date format for the DI conversion code. The value should be a D conversion code. The order is specified by the DMY order (field 23). The separator is specified by the date separator (field 24).
7	Time 'MT' format	The default time format for the MT conversion code. The value should be an MT conversion code. In most cases, use the value TI.
8	Time 'TI' format	The format for the TI conversion code. The value should be an MT conversion code that specifies separators. The default separator is a colon (:) as specified by the time separator (field 25).
9	Days of the week	A multivalued list of the full names of the days of the week. For example, Monday, Tuesday. Fields 9 and 10 are associated multi-valued fields; the same number of values must exist in each field.
10	Abbreviated	A multivalued list of abbreviated names of the days of the week. For example, Mon, Tue. See field 9.
11	Month names	A multivalued list of the full names of the months of the year. For example, January, February. Fields 11 and 12 are associated multivalued fields; the same number of values must exist in each field.
12	Abbreviated	A multivalued list of abbreviated names of the months of the year. For example, Jan, Feb. See field 11.

NLS.LC.TIME File Attributes (Continued)

Field	Name	Description
13	Chinese years	A multivalued list of Chinese year names (Monkey to Sheep).
14	AM string	A string used to denote times before noon in 12-hour formats.
15	PM string	A string used to denote times after noon in 12-hour formats.
16	BC string	A string to be added to dates before the date 01 Jan 0001 in the Gregorian calendar. This corresponds to -718432, the UniVerse internal date.
17	Era name	A multivalued list of names of eras and their start dates, beginning with the most recent, for example, Japanese Imperial Era Heisei. This field can be used for any locale that uses a calendar with several year zeros. For example, the Thai Buddhist Era commencing 1/1/543 BC. See “ Defining Era Names ” on page 11.
18	Start date	Corresponding era start dates for the era names specified in UniVerse internal date format.
19	HEADING/FOOTING D format	A D or DI conversion code used in HEADING and FOOTING statements.
20	HEADING/FOOTING T format	An MT or TI conversion code used in HEADING and FOOTING statements.
21	Gregorian calendar day 1	The date at which the calendar changes from Julian to Gregorian, expressed as a UniVerse internal date. The default is -140607, corresponding to 11 January 1583.
22	Number of days skipped	The number of days to skip when the calendar changes from Julian to Gregorian. The default is 10.

NLS.LC.TIME File Attributes (Continued)

Field	Name	Description
23	Default DMY order	The order of day, month, and year, for example, DMY.
24	Default date separator	The separator used between day, month, and year. The default is the slash (/).
25	Default time separator	The separator used between hours, minutes, and seconds. The default is the colon (:).

NLS.LC.TIME File Attributes (Continued)

Defining Era Names

The values in the ERA_NAMES field can contain the format code:

Name [%*n*] [*string*]

Name is the era name.

%*n* is a digit from 1 through 9, or the characters +, -, or Y.

string is any text string.

The %*n* syntax allows era year numbers to be included in the era name and indicates how the era year numbers are to be calculated. If %*n* is omitted, %1 is assumed.

The rules for the %*n* syntax are as follows:

%1 – %9: The number following the % is the number to be used for the first year *n* this era. This is effectively an offset which is added to the era year number. This will usually be 1 or 2.

%+: The era year numbers count backward relative to year numbers, that is, if era year number 1 corresponds to Julian year Y, year 2 corresponds to Y–1, year 3 to Y–2, and so forth.

%–: The same as %+, but uses negative era year numbers, that is, first year Y is –1, Y–1 is –2, Y–2 is –3, and so forth.

%Y: Uses the Julian year numbers for the era year numbers. The year number will be displayed as a 4-digit year number.

The %+, %-, and %Y syntax should only be used in the last era name in the list of era names, that is, the first era, since the list of era names must be in descending date order.

string allows any text string to be appended to the era name. It is frequently the case that the first year or part-year of an era is followed by some qualifying characters. Therefore, the actual era is divided into two values, each with the same era name, but one terminated by %1*string* and the other by %2. You must define the era names accordingly.

Example

This example shows the contents of the records named DEFAULT and US-ENGLISH in the NLS.LC.TIME file. The US-ENGLISH record is based on the ENGLISH.NAMES record. An empty field specifies that its definition is derived from any category on which it is based. If there is no base category, the default category is used.

```

Time/Date Conventions for Locale DEFAULT

Category name..... DEFAULT
Description..... System defaults
Based on.....
TIMEDATE format..... MTS
                   . D4
Full DATE format..... D4WAMADY[" ", " ", " ", " ", " "]
                   . MT
Date 'D' format..... D4 DMBY
Date 'DI' format..... D2-YMD
Time 'MT' format..... TI
Time 'TI' format..... MTS:
Days of the week.....
Abbreviated.....
Sunday                               Sun
Monday                               Mon
Tuesday                              Tue
Wednesday                            Wed
Thursday                             Thu
Friday                               Fri
Saturday                             Sat
Month names.....
Abbreviated.....
January                              Jan
February                             Feb
March                                 Mar
April                                 Apr
May                                   May
June                                  Jun
July                                  Jul

```

```

August
September
October
November
December
Chinese years..... MONKEY
                    . COCK
                    . DOG
                    . BOAR
                    . RAT
                    . OX
                    . TIGER
                    . RABBIT
                    . DRAGON
                    . SNAKE
                    . HORSE
                    . SHEEP
AM string..... am
PM string..... pm
BC string..... BC
Era name..... Start date....
Heisi                08 JAN 1989
Showa                25 DEC 1926
Taisho               30 JUL 1912
Meiji                08 SEP 1868
HEADING/FOOTING D format. D2-
HEADING/FOOTING T format. MTS
                    . D2-
Gregorian calendar day 1.    11 JAN 1583
Number of days skipped...    10
Default DMY order.....
Default date separator...
Default time separator...
Time/Date Conventions for US-ENGLISH

Category name..... US-ENGLISH
Description..... Territory=USA, Language=English
Based on..... .ENGLISH.NAMES
TIMEDATE format.....
Full DATE format.....
Date 'D' format.....
Date 'DI' format..... D2/MDY
Time 'MT' format.....
Time 'TI' format..... MTHS:
Days of the week.....Abbreviated.....

Month names..... Abbreviated.....

Chinese years.....
AM string.....
PM string.....
BC string.....
Era name..... Start date....

```

```

HEADING/FOOTING D format.
HEADING/FOOTING T format.
Gregorian calendar day 1.
Number of days skipped...
Default DMY order..... MDY
Default date separator...
Default time separator...

```

Numeric Records

Convention records in the Numeric category are stored in the NLS.LC.NUMERIC file. The following table shows each field number, its display name, and a description:

Field	Name	Description
0	Category Name	The name of the convention.
1	Description	A description of the convention. It usually includes the territory that the convention applies to and the language it is used with.
2	Based on	The name of another convention record in the NLS.LC.NUMERIC file that this convention is based on.
3	Decimal separator	The character used as a decimal separator (radix character). The value can be expressed as either a single character or the hexadecimal Unicode value of a character.
4	Thousands separator	The character used as a thousands separator. The value can be expressed as either a single character or the hexadecimal Unicode value of a character. Use the value NONE to indicate that no separator is needed.
5	Suppress leading zero	Defines whether leading zeros should be suppressed for numbers in the range 1 through -1. A value of 0 or N means insert a zero; any other value suppresses the zero.
6	Alternative digits (0 first)	A multivalued field containing 10 values that can be used as alternatives to the corresponding ASCII digits 0 through 9.

NLS.LC.NUMERIC File Attributes

This example shows the contents of the records named DEFAULT and DEC.COMMA+DOT locale (used by DE-GERMAN) in the NLS.LC.NUMERIC file. The DEC.COMMA+DOT conventions are based on DEFAULT.

Numeric Conventions for DEFAULT

```

Category name..... DEFAULT
Description..... System defaults: Decimal separator =
                    dot, thousands = comma

Based on.....
Decimal separator..... - FULL STOP
Thousands separator... - COMMA
Suppress leading zero. 0
Alternative digits (0 first).

```

Numeric Conventions for DEC.COMMA+DOT

```

Category name..... DEC.COMMA+DOT
Description..... Decimal separator = comma, thousands =
                    dot
Based on..... DEFAULT
Decimal separator..... , - COMMA
Thousands separator... - FULL STOP
Suppress leading zero.
Alternative digits (0 first).

```

Monetary Records

Convention records in the Monetary category are stored in the NLS.LC.MONETARY file. The following table shows each field number, its display name, and a description:

Field	Name	Description
0	Category Name	The name of the convention.
1	Description	A description of the convention. It usually includes the territory that the convention applies to and the language it is used with.
2	Based on	The name of another convention record in the NLS.LC.MONETARY file on which this category is based.

NLS.LC.MONETARY File Attributes

Field	Name	Description
3	Monetary decimal separator	The character used as a decimal separator (radix character). You do not need to specify a value if this character is the same as the one in the decimal separator field in the corresponding convention in NLS.LC.NUMERIC.
4	Monetary thousands separator	The character used as a thousands separator. You do not need to specify a value if this character is the same as the one in the thousands separator field in the corresponding convention in the NLS.LC.NUMERIC file.
5	Local currency symbol	A character or string used as the local currency symbol, for example, \$ or ¥. Leading or trailing spaces are not included.
6	International currency symbol	The international currency symbol. The value should consist of three uppercase ASCII characters as specified in the ISO 4217 standard. For example, USD. Trailing spaces are included. This symbol always precedes the amount it refers to.
7	Decimal places	The number of decimal places in monetary amounts when used with the local currency symbol is used.
8	International decimal places	The number of decimal places in monetary amounts when used with the international currency symbol (field 6).
9	Positive sign	The sign used to indicate positive monetary amounts. If the value consists of two characters, these are used to parenthesize positive monetary amounts (one used at either end of the monetary format). Use the value NONE to omit a positive sign.

NLS.LC.MONETARY File Attributes (Continued)

Field	Name	Description
10	Negative sign	The sign used to indicate negative monetary amounts. If the value consists of two characters, these are used to parenthesize negative monetary amounts (one used at either end of the monetary format). Use the value NONE to omit a negative sign.
11	Positive currency format	The format for positive monetary amounts. This is expressed using a combination of the characters \$ S + 1 and a space. The \$ or S represents the local currency symbol. 1 represents the monetary amount. + represents the positive sign. If the positive sign (field 9) contains two characters, the + sign is ignored. For example, the value \$1 in a US locale results in the format \$1,234.56. The value 1 \$ in a GERMAN locale results in the format 1.234,56 DM.
12	Negative currency format	The format for negative monetary amounts. This is expressed using a combination of the characters \$ S - 1 and a space. The \$ or S represents the local currency symbol. 1 represents the monetary amount. - represents the negative sign. If the negative sign (field 10) contains two characters the - sign is ignored. For example, the value -\$1 in a PORTUGUESE locale results in the format -1,234\$56. The value \$ -1 in a DUTCH locale results in the format F -1.234,56.

NLS.LC.MONETARY File Attributes (Continued)

This example shows the contents of the record named DEFAULT in NLS.LC.MONETARY, followed by records for NETHERLANDS, ITALY, NORWAY and PORTUGAL, which show different combinations of fields:

```

Numeric Conventions for DEFAULT

Category name..... DEFAULT
Description..... System defaults
Based on.....
Monetary decimal separator... . - FULL STOP
Monetary thousands separator.. , - COMMA
Local currency symbol..... $ - DOLLAR SIGN
International currency symbol. USD<SP>

```

Decimal places..... 2
 International decimal places.. 2
 Positive sign..... NONE
 Negative sign..... - - HYPHEN-MINUS
 Positive currency format..... S1
 Negative currency format..... S-1

Monetary Conventions for NETHERLANDS

Category name..... NETHERLANDS
 Description..... Territory=Netherlands
 Based on.....
 Monetary decimal separator... , - COMMA
 Monetary thousands separator.. . - FULL STOP
 Local currency symbol..... F1
 International currency symbol. NLG<SP>
 Decimal places..... 2
 International decimal places.. 2
 Positive sign..... NONE
 Negative sign..... - - HYPHEN-MINUS
 Positive currency format..... S 1
 Negative currency format..... S 1-

Monetary Conventions for ITALY

Category name..... ITALY
 Description..... Territory=Italy
 Based on.....
 Monetary decimal separator... , - COMMA
 Monetary thousands separator.. . - FULL STOP
 Local currency symbol..... L.
 International currency symbol. ITL.
 Decimal places..... 0
 International decimal places.. 2
 Positive sign..... NONE
 Negative sign..... - - HYPHEN-MINUS
 Positive currency format..... S1
 Negative currency format..... -S1

Monetary Conventions for NORWAY

Category name..... NORWAY
 Description..... Territory=Norway
 Based on.....
 Monetary decimal separator... , - COMMA
 Monetary thousands separator.. . - FULL STOP
 Local currency symbol..... kr
 International currency symbol. NOK<SP>
 Decimal places..... 2
 International decimal places.. 2
 Positive sign..... NONE
 Negative sign..... - - HYPHEN-MINUS
 Positive currency format..... S1


```

Negative currency format..... S1-

Monetary Conventions for PORTUGAL

Category name..... PORTUGAL
Based on Category name..... PORTUGAL
Description..... Territory=Portugal
Based on.....
Monetary decimal separator... $      - DOLLAR SIGN
Monetary thousands separator.. .    - FULL STOP
Local currency symbol..... NONE
International currency symbol. PTE<SP>
Decimal places..... 2
International decimal places.. 2
Positive sign..... NONE
Negative sign..... -      - HYPHEN-MINUS
Positive currency format..... 1 S
Negative currency format..... -1 S

```

The following table shows how the data in the previous records affect monetary formats:

Locale Name	Positive Format	Negative Format	International Format
DEFAULT	\$1,234.56	\$-1,234.56	USD 1,234.56
NETHERLANDS	Fl 1.234,56	Fl 1.234,56-	NLG 1.234,56
ITALY (see Note)	L.1.234	-L.1.234	ITL.1.234
NORWAY	kr1.234,56	kr1.234,56-	NOK 1.234,56
PORTUGAL	1.234\$56	-1.234\$56	PTE 1,234\$56

Monetary Format Examples

Note: Italian lire are usually quoted in whole numbers only. Your programs must detect that the DEC_PLACES and INTL_DEC_PLACES fields contain zero in this case, and not hard code an MD2 conversion. An MM conversion handles the scaling automatically.

Ctype Records

Convention records in the Ctype category are stored in the NLS.LC.CTYPE file. The following table shows each field number, its display name, and a description.





Note: For fields 3 onward, you can enter the values as characters or as Unicode values. You can specify a range of values separated by a dash (-).

Field	Name	Description
0	Category Name	The name of the convention.
1	Description	A description of the convention. It usually includes the territory to which the convention applies and the language with which it is used.
2	Based on	The name of another convention record in the NLS.LC.CTYPE file on which this convention is based.
3	Lowercase	A multivalued list of lowercase values whose associated uppercase values differ from the defaults in NLS.CS.CASES.
4	->Upper	A multivalued list of the equivalent uppercase values for the characters listed in field 3.
5	Uppercase	A multivalued list of uppercase values whose associated lowercase values differ from the defaults in NLS.CS.CASES.
6	->Lower	A multivalued list of the equivalent lowercase values for the characters listed in field 5.
7	Alphabetic	A multivalued list of characters that are alphabetic, but are not described as such in the NLS.CS.ALPHAS file. You can specify this value as a Unicode block value using the format BLOCK= <i>nn</i> , where <i>nn</i> is the Unicode block number.
8	Non-Alphabetic	A multivalued list of characters that are not alphabetic, but are described as such in the NLS.CS.ALPHAS file. You can specify this value as a Unicode block value using the format BLOCK= <i>nn</i> , where <i>nn</i> is the Unicode block number.
9	Numerics	A multivalued list of characters that should be considered as numeric but are not described as such in the NLS.CS.TYPES file.
10	Non-Numerics	A multivalued list of characters that are not considered to be numeric but are described as such in the NLS.CS.TYPES file.

NLS.LC.CTYPE File Attributes

Field	Name	Description
11	Printables	A multivalued list of characters that are considered to be printable but are not described as such in the NLS.CS.TYPES file.
12	Non-Printables	A multivalued list of characters that are not considered to be printable but are described as such in the NLS.CS.TYPES file.
13	Trimables	A multivalued list of characters that are to be removed by TRIM functions in addition to spaces and tab characters.

NLS.LC.CTYPE File Attributes (Continued)

In Spanish, accented characters other than ñ drop their accents when converted to uppercase. In French, all accented characters drop their accents in uppercase.

This example shows a convention called NOACCENT.UPCASE, which the locale FR-FRENCH uses, and a convention called SPANISH, that is based on it.



Note: In this example, the only characters affected are those in general use in French and Spanish. There are many other accented characters in Unicode. This example displays <N?> that comes from the MNEMONICS map. This lets you easily enter non-ASCII characters rather than their Unicode values.

```

Character Type Conventions for ACCENTLESS.UPPERCASE

Category name. NOACCENT.UPCASE
Description... ISO8859-1 lowercase accented chars lose
                accents in uppercase
Based on..... DEFAULT
Lowercase..... ->
Uppercase.....
00E0 - LATIN SMALL LETTER A WITH GRAVE  0041 - LATIN CAPITAL LETTER A
00E1 - LATIN SMALL LETTER A WITH ACUTE  0041 - LATIN CAPITAL LETTER A
00E2 - LATIN SMALL LETTER A WITH
        CIRCUMFLEX                      0041 - LATIN CAPITAL LETTER A
00E3 - LATIN SMALL LETTER A WITH TILDE  0041 - LATIN CAPITAL LETTER A
00E4 - LATIN SMALL LETTER A WITH
        DIAERESIS                        0041 - LATIN CAPITAL LETTER A
00E5 - LATIN SMALL LETTER A WITH RING
        ABOVE                             0041 - LATIN CAPITAL LETTER A
00E7 - LATIN SMALL LETTER C WITH
        CEDILLA                          0043 - LATIN CAPITAL LETTER C
00E8 - LATIN SMALL LETTER E WITH GRAVE  0045 - LATIN CAPITAL LETTER E
00E9 - LATIN SMALL LETTER E WITH ACUTE  0045 - LATIN CAPITAL LETTER E
00EA - LATIN SMALL LETTER E WITH
        CIRCUMFLEX                      0045 - LATIN CAPITAL LETTER E
00EB - LATIN SMALL LETTER E WITH
        DIAERESIS                        0045 - LATIN CAPITAL LETTER E
00EC - LATIN SMALL LETTER I WITH GRAVE  0049 - LATIN CAPITAL LETTER I
00ED - LATIN SMALL LETTER I WITH ACUTE  0049 - LATIN CAPITAL LETTER I

```

00EE - LATIN SMALL LETTER I WITH CIRCUMFLEX	0049 - LATIN CAPITAL LETTER I
00EF - LATIN SMALL LETTER I WITH DIAERESIS	0049 - LATIN CAPITAL LETTER I
00F1 - LATIN SMALL LETTER N WITH TILDE	004E - LATIN CAPITAL LETTER N
00F2 - LATIN SMALL LETTER O WITH GRAVE	004F - LATIN CAPITAL LETTER O
00F3 - LATIN SMALL LETTER O WITH ACUTE	004F - LATIN CAPITAL LETTER O
00F4 - LATIN SMALL LETTER O WITH CIRCUMFLEX	004F - LATIN CAPITAL LETTER O
00F5 - LATIN SMALL LETTER O WITH TILDE	004F - LATIN CAPITAL LETTER O
00F6 - LATIN SMALL LETTER O WITH DIAERESIS	004F - LATIN CAPITAL LETTER O
00F8 - LATIN SMALL LETTER O WITH STROKE	004F - LATIN CAPITAL LETTER O
00F9 - LATIN SMALL LETTER U WITH GRAVE	0055 - LATIN CAPITAL LETTER U
00FA - LATIN SMALL LETTER U WITH ACUTE	0055 - LATIN CAPITAL LETTER U
00FB - LATIN SMALL LETTER U WITH CIRCUMFLEX	0055 - LATIN CAPITAL LETTER U
00FC - LATIN SMALL LETTER U WITH DIAERESIS	0055 - LATIN CAPITAL LETTER U
00FD - LATIN SMALL LETTER Y WITH ACUTE	0059 - LATIN CAPITAL LETTER Y
00FF - LATIN SMALL LETTER Y WITH DIAERESIS	0059 - LATIN CAPITAL LETTER Y

Uppercase..... -> Lowercase.....

Alphabetic.....
 Non-Alphabetic..
 Numeric.....
 Non-Numeric....
 Printable.....
 Non-Printable..
 Trimmable.....

Character Type Conventions for SPANISH

Category name. SPANISH

Description... Language=Spanish - SMALL N WITH TILDE
 keeps tilde on uppercasing

Based on..... NOACCENT.UPCASE

Lowercase..... ->

Uppercase.....

<n?> - LATIN SMALL LETTER N WITH TILDE <N?> - LATIN CAPITAL LETTER N WITH TILDE

Uppercase..... ->

Lowercase.....

Alphabetic.....
 Non-Alphabetic..
 Numeric.....
 Non-Numeric....
 Printable.....
 Non-Printable..
 Trimmable.....

Collate Records

Convention records in the Collate category are stored in the NLS.LC.COLLATE file. The following table shows each field number, its display name, and a description. Many of the fields are Boolean. An empty field or a value of 0 or N indicates false; any other value indicates true.

Field	Name	Description
0	Category Name	The name of the convention.
1	Description	A description of the convention. It usually includes the territory to which the convention applies and the language with which it is used.
2	Based on	The name of another convention record in the NLS.LC.COLLATE file on which this convention is based.
3	Accented Sort?	This field determines how accents on characters affect the collate order. A false value indicates that accents are not collated separately. A true value indicates that accents are used as tie breakers in the sort. See “Collating” on page 28.
4	In reverse?	If field 3 indicates an accented collation, this field determines the direction of that collation. A false value indicates forward collation. A true value indicates reverse collation.
5	Cased Sort?	This field determines whether the case of a character is considered during collation. A false value indicates that case is not considered. A true value indicates that case is used as a tie breaker in the collation.
6	Lowercase first?	If field 5 indicates a cased collation, this field determines which case is collated first. A false value indicates that lowercase is collated first. A true value indicates that uppercase is collated first.
7	Expand	A multivalued field containing Unicode values of characters that are expanded before collation. See “Contractions and Expansions” on page 31.

NLS.LC.COLLATE File Attributes

Field	Name	Description
8	Expanded	A multivalued field associated with field 7 that supplies the values to which the characters expand. Each value may be one or more Unicode values separated by tab characters or spaces. To override an expansion inherited from a based convention named in field 2, enter the same multivalued value in fields 7 and 8. (For another method, see the description of field 10.)
9	Before?	A multivalued field associated with fields 7 and 8 that determines how expanded characters collate. A false value indicates that a character is collated after expansion; a true value indicates that a character is collated before expansion.
10	Contract	A multivalued field containing a list of pairs of Unicode values of characters after contraction. The values should be separated by tab characters or spaces. To override an expansion inherited from a based convention named in field 2, enter a value in this field and a corresponding empty value in field 11. See “ Contractions and Expansions ” on page 31.
11	Before	A multivalued field associated with field 10. It gives the Unicode value of the character that a contracted pair precedes in the collation order.
12	Weight Tables	A multivalued field supplying the weight information for characters in this locale. The values should be record IDs in the NLS.WT.TABLES file. The default is the name of the locale. The weight information is processed in the order supplied in this field.

NLS.LC.COLLATE File Attributes (Continued)

This example shows the NLS.LC.COLLATE records named DEFAULT, GERMAN, and SPANISH:

- DEFAULT uses no expansion or contraction, but does collate in a sequence other than the Unicode value.
- GERMAN uses the DEFAULT collating sequence, but introduces an expansion.

■ SPANISH is also based on DEFAULT, but introduces eight contractions.

Collating Sequence Conventions for DEFAULT

```

Category name... DEFAULT
Description..... System defaults
Based on.....
Accented Sort?... N
In reverse?..... N
Cased Sort?..... N
Lowercase first?. N
Expand ----->..... Before? Expanded..
.....

Contract... ----->..... Before
.....

Weight Tables... LATIN1-DEFAULT
                  . LATINX-DEFAULT
                  . LATINX2-DEFAULT
                  . LATINX3-DEFAULT
                  . GREEK-DEFAULT
                  . CYRILLIC-DEFAULT

```

Collating Sequence Conventions for GERMAN

```

Category name... GERMAN
Description..... Language=German
Based on..... DEFAULT
Accented Sort?... Y
In reverse?..... N
Cased Sort?..... Y
Lowercase first?. N
Expand ----->..... Before? Expanded..
.....
<ss>  LATIN SMALL LETTER SHARP S      N S S      LATIN CAPITAL LETTER S
                                           LATIN CAPITAL LETTER S

Contract... ----->..... Before
.....

Weight Tables....

```

Collating Sequence Conventions for SPANISH

```

Category name... SPANISH
Description..... Language=Spanish
Based on..... DEFAULT
Accented Sort?... Y
In reverse?..... N
Cased Sort?..... Y
Lowercase first?. N
Expand ----->..... Before? Expanded..
.....

Contract... ----->..... Before
.....
C H          LATIN CAPITAL LETTER C      D      LATIN CAPITAL LETTER D
              LATIN CAPITAL LETTER H
C h          LATIN CAPITAL LETTER C      D      LATIN CAPITAL LETTER D

```

c h	LATIN SMALL LETTER C	d	LATIN SMALL LETTER D
	LATIN SMALL LETTER H		
c H	LATIN SMALL LETTER C	d	LATIN SMALL LETTER D
	LATIN CAPITAL LETTER H		
L L	LATIN CAPITAL LETTER L	M	LATIN CAPITAL LETTER M
	LATIN CAPITAL LETTER L		
L l	LATIN CAPITAL LETTER L	M	LATIN CAPITAL LETTER M
	LATIN SMALL LETTER L		
l l	LATIN SMALL LETTER L	m	LATIN SMALL LETTER M
	LATIN SMALL LETTER L		
l L	LATIN SMALL LETTER L	m	LATIN SMALL LETTER M
	LATIN CAPITAL LETTER L		
Weight Tables....	LATIN-SPANISH		

Collating

Collating is a complex issue for many languages. It is not sufficient to collate a character set in numerical order of its Unicode values. Locales that share a character set often have different collating rules. For example, these are the main issues that affect collating in Western European languages:

- Accented characters. Should accented characters come before or after their unaccented equivalents? Or should accents only be examined if two strings being compared would otherwise be identical (that is, as a tie breaker)?
- Expanding characters. Some languages treat certain single characters as two separate characters for collating purposes.
- Contracting characters. Some languages have pairs of characters that collate as though they were a single character.
- Should case be considered? Should case be used as a tie breaker for otherwise identical strings? If so, which comes first, uppercase or lowercase?
- Should hyphens or other punctuation be considered as tie breakers?

How UniVerse Collates

To overcome these collating problems, UniVerse allows each Unicode character to be assigned up to three weights. The weight is a numeric value to use instead of the character during collation. The three weights are as follows:

Weight	Description
Shared weight	All characters that are essentially the same have the same shared weight, even though they may differ in accent or case.
Accent weight	This weight shows the order of precedence for accented characters. The Collate convention determines the <i>direction</i> of the collation.
Case weight	This weight differentiates between uppercase and lowercase characters. The Collate convention determines which case has precedence.

Unicode Character Weights

Before collation begins, UniVerse expands or contracts any characters as defined in the Collate convention. The collation works as follows:

1. The characters are compared by shared weight.
2. If two characters have the same shared weight, they are compared by accent weight.
3. If the accent weight is the same, they are compared by case weight.

Example of Accented Collation

This table compares how four French words that differ only in their accents are collated in two different ways, depending on how the weight tables have been configured:

Order	Accented Collation	Unaccented Collation
1	cote	cote
2	côte	coté
3	coté	côte
4	côté	côté

Accented Collation Examples

In the accented collation, the words are in the order they would be found in a French dictionary. (It is actually a reverse accented collation.) Each accented character has the same shared weight as it would have without the accent. The order is decided by referring to the accent weight.

In the unaccented collation, each accented character has a different shared weight unrelated to its unaccented equivalent. The order is decided by the shared weight alone.

Example of Cased Collation

The three words Aaron, Aardvark, and aardvark show how case affects collation:

Order	Cased Collation	Uncased Collation
1	Aardvark	Aardvark
2	aardvark	Aaron
3	Aaron	aardvark

Cased Collation Example

In the cased collation, Aaron follows aardvark because the characters ‘A’ and ‘a’ have the same shared weight. The case weight is only considered for the two strings that are otherwise identical, that is, Aardvark and aardvark.

In the uncased collation, Aaron precedes aardvark because the characters ‘A’ and ‘a’ have different shared weights.

Shared Weights and Blocks

Unicode is divided into blocks of related characters. For example, Cyrillic characters form one block, while Hebrew characters form another. In most circumstances, it is unlikely that you need to collate characters from more than one block at a time. Shared weights are assigned so that characters collate correctly within each Unicode block.

Contractions and Expansions

Some languages have pairs of characters that collate as though they were a single character. Other languages treat certain single characters as two separate characters for collating. These contractions and expansions are done before UniVerse begins a collation.

For example, in Spanish, the character pairs CH and LL (in any combination of case) are treated as a single, separate character. CH comes between C and D in the collating sequence, and LL comes between L and M. UniVerse identifies these character pairs before collation begins. In German, the character ß is expanded to SS before collation begins.

Editing Weight Tables

Collating character sets in different languages is a complex issue. Each character has an assigned weight value used for numeric comparisons in sorting, but you can change these weight values to sort in a different way when you want to customize your locale.

You can edit the weight table for a locale by choosing Categories -> Weight Tables -> Edit from the NLS Administration menu. Any change you make to the weight assigned to a character overrides the default weight derived from its Unicode value.

The weights are held in the NLS.WT.TABLES file, which is a type 19 file. Each record in the file can contain:

- Comment lines, introduced by a # or *
- A set of weight values for a Unicode code point

Each weight value line has the following fields, separated by at least one ASCII space or tab character:

```
character [block.weight / ] shared.weight accent.weight case.weight  
[comments]
```

character is a Unicode character value. This should be four hexadecimal digits, zero-filled as necessary.

The *block.weight* / *shared.weight* value is one or two decimal integers, separated by a slash (/) if necessary. *block.weight* can be 1 through 127; *shared.weight* 1 through 32767. If *block.weight* is omitted, it is taken as the value of the Unicode block number to which *character* belongs. *shared.weight* may be given as a hyphen, in which case it is taken as the value of the most recent weight value line without a hyphen for *shared.weight*. Characters that should sort together if accents and case are disregarded should have the same *block.weight* / *shared.weight* value.

accent.weight is a decimal integer 1 through 63. It may be given as a hyphen, in which case it is taken as the value of the most recent weight value line without a hyphen for *accent.weight*. Characters that are distinguished only by accent should have the same *block.weight* / *shared.weight* value and differ in their *accent.weight* value. A list of conventional values to assign to this field can be found by listing records starting with "AW..." in the NLS.WT.LOOKUP file.

case.weight value is a decimal integer 1 through 7, or the letter U or L to indicate uppercase and lowercase. *case.weight* can be given as a hyphen, in which case it is taken as the value of the most recent weight value line without a hyphen for *case.weight*. Characters that are distinguished only by case should have the same *block.weight* / *shared.weight* value and *accent.weight* value and differ only in their *case.weight* value. A list of conventional values to assign to this field can be found by listing records starting with “CW...” in the NLS.WT.LOOKUP file.

comments can contain any characters.

Calculating the Overall Weight

The overall weight assigned to *character* is calculated using the following formula:

$$(block.weight \times 2^{24}) + (shared.weight \times 2^9) + (accent.weight \times 2^3) + case.weight$$

If *character* is not mentioned in a table, the default weight is calculated as follows:

$$(BW \times 2^{24}) + (SW \times 2^9)$$

BW is the character’s Unicode block number. SW depends on its position within the block: the first character has a SW of 1, the second a SW of 2, and so on.

Example of a Weight Table

This example shows a weight table for collating Turkish characters:

```
* Sorting weight table for TURKISH characters (from ISO8859/9)
* in order on top of LATIN1/LATINX tables. These characters are:
*
* Between G and H: G BREVE
* Between H and J: I WITH DOT ABOVE (uppercase version of SMALL I
0069)
* DOTLESS I (lowercase version of CAPITAL I 0049)
* (Note: the sequence is H, dotless I, I dot + accented versions,
J, ...)
* Between S and T: S CEDILLA
*
* SYNTAX:
* Each non-comment line gives one or more weights for a
character, as * follows (character value in hex, weights in
decimal):
* Field 1 = Unicode character value
* Field 2 = Shared weight (characters that sort together if
accents and case were to be disregarded should
```

```

*           have the same SW)
*       Or, Block Weight/Shared Weight. This form allows
*       characters in different Unicode blocks to have
*       equal SWs. If BW is omitted, only SWs for characters
in *           the same block are equal.
* Field 3 = Accent weight, or '-' to omit or copy from previous.
*           Please use values as defined in the file
NLS.WT.LOOKUP.
* Field 4 = Case weight, or 'U' for upper and 'L' for lower case
chars.
*
*****
* HEX      (BW/)SW          AW      CW
* After G:
011E      4/1092           5        U * G WITH BREVE
011F      -                5        L
* I, dotted and undotted:
* (Note we do not use AWs here, but use SWs to differentiate
* these characters from the unaccented versions.)
0049      4/1109           -        U * I
0131      -                -        L * DOTLESS I
0130      4/1110           -        U * I WITH DOT ABOVE
0069      -                -        L * I
* S cedilla
015E      4/1232           40     U * S WITH CEDILLA
015F      -                40     L
*
* END

```

Using Locales

From within a UniVerse BASIC program you can do the following:

- Retrieve the current locale name of a specified category
- Save the current locale settings
- Restore the saved locale settings
- List the current locale settings
- Change the current locale settings

For information about using functions to do these tasks from within BASIC programs, see Chapter 5, [“NLS in UniVerse BASIC Programs.”](#)

Retrieving Locale Settings

You can retrieve locale settings in two ways:

- From the UniVerse prompt using the GET.LOCALE command
- From a UniVerse BASIC program using the GETLOCALE or LOCALEINFO functions (see Chapter 5, [“NLS in UniVerse BASIC Programs”](#)).

GET.LOCALE displays the locale names set in each category, and details of any saved locale, if it differs from the current one. If locales are not enabled on the system, or if NLS mode is off, GET.LOCALE returns an error.

Saving and Restoring Locales

You can save and restore locales in two ways:

- From the UniVerse prompt using the SAVE.LOCALE and RESTORE.LOCALE commands.
- From a UniVerse BASIC program using the SETLOCALE function. This is described in detail in [Changing the Current Locale](#) in Chapter 5, [“NLS in UniVerse BASIC Programs.”](#)

A locale is always set up and saved when you enter UniVerse. You can restore this initial locale using `RESTORE.LOCALE` if you have not issued a `SAVE.LOCALE` command during your UniVerse session. `SAVE.LOCALE` and `RESTORE.LOCALE` return errors if they are issued when locales are turned off, that is, if either the `NLSLCMODE` or `NLSMODE` configurable parameters in the `uvconfig` file is set to 0.

Listing Current Locales

You can list the current locales from the UniVerse prompt using the `LIST.LOCALES` command. The `LIST.LOCALES` command uses an existing active select list; otherwise, it lists all installed locales.

Changing Current Locales

You can change or disable locale settings in two ways:

- From the UniVerse prompt using the `SET.LOCALE` command
- From a UniVerse BASIC program using the `SETLOCALE` function (see Chapter 5, “[NLS in UniVerse BASIC Programs](#)”).

You can disable a locale or set a new locale from the UniVerse prompt using the `SET.LOCALE` command. `SET.LOCALE` returns an error if locales are not enabled, that is, if either the `NLSLCMODE` or the `NLSMODE` configurable parameter is set to 0.



***Note:** When you want to specify numeric and monetary formatting for a locale, you must set both the Numeric and Monetary categories to something other than `OFF`, for example, `DEFAULT`. If not, UniVerse treats BASIC conversions, such as `MD`, `ML`, and `MR`, as if locales are turned off.*

NLS in UniVerse BASIC Programs

How UniVerse BASIC Is Affected	5-4
Using the UVNLS.H Include File	5-4
String Length	5-5
Length of Record IDs	5-6
Display Length in BASIC	5-7
Finding the Display Length of a String	5-7
Formatting a String in Display Positions	5-7
Folding Strings Using Display Positions	5-8
Inputting Using Display Length with INPUTDP	5-8
Block Size Always in Bytes	5-8
The REMOVE Pointer and Multibyte Character Sets	5-9
Maps in UniVerse BASIC	5-10
Determining a File's Map Name	5-10
Maps for Source Files	5-11
Maps and Devices	5-12
Maps for Auxiliary Devices	5-12
@ Function Codes for Terminal and Auxiliary Maps	5-12
Printing Previously Mapped Data with UPRINT	5-13
Finding the Map Associated with a Print Channel	5-13
Maps for UNIX Pipes	5-13
Unmappable Characters	5-15
Unmappable Characters and WRITE Statements	5-15
Unmappable Characters and READ Statements	5-16
Multinational Characters in UniVerse BASIC	5-18
Editing Multinational Characters	5-18
Generating Characters in External Format	5-20
Generating Characters in Internal Format	5-21

Internal and External String Conversion	5-21
Displaying Records by Character Value	5-24
Exchanging Character Values	5-25
Case Inversion and Deadkey Characters	5-25
BASIC and Locales	5-26
Retrieving Locale Settings	5-26
Saving and Restoring Locales	5-26
Changing the Current Locale.	5-26

This chapter describes how UniVerse BASIC programs use NLS. The topics covered include:

- How UniVerse BASIC is affected by NLS.
- Display length in UniVerse BASIC. This describes how to accommodate the difference between a character's display length and its string length.
- Maps in UniVerse BASIC. This covers how maps are used by files and devices, how to set and modify maps, and how UniVerse BASIC handles unmappable characters.
- Multinational characters in UniVerse BASIC. This describes how you can include multinational characters in source code, specify them for printing, or edit them using ED.
- Using locales in UniVerse BASIC. This topic describes how to set or query a locale from within a program.

How UniVerse BASIC Is Affected

UniVerse BASIC is aware of multinational characters and locales. Usually this is transparent to the programmer and no special code is needed. There is usually no need to recompile existing programs for NLS. If you write programs that use NLS features such as locales, you should compile the programs with UniVerse in NLS mode. Any program that uses NLS features should be run with UniVerse in NLS mode, otherwise you may see run-time errors.

UniVerse BASIC is fundamentally unchanged by NLS, except for some new or modified UniVerse BASIC statements and functions. UniVerse BASIC statement and variable names must be in ASCII with the exception of comments and literal strings. For more information about when you can use ASCII and non-ASCII characters, see [“Multinational Characters in BASIC”](#) on page 18.

Using the UVNLS.H Include File

You can use the SYSTEM function to test whether NLS mode is on when a program runs, and to extract information about NLS settings. The following system function values are read-only. Their tokens are in the include file UVNLS.H.

Value	Token	Return Value
100	NLS\$ON	1 if NLS is installed and NLSMODE is on, otherwise 0. Use this value to check if NLS maps are enabled.
101	NLS\$LOCALES	The value of the NLSLCMODE parameter, otherwise 0. Use this value to check if NLS locales are enabled.
102	NLS\$MESSAGES	Reserved for future enhancements. Always returns 0.
103	NLS\$TERMMAP	The terminal map name assigned to the current terminal print channel, otherwise 0.
104	NLS\$AUXMAP	The auxiliary printer map name assigned to the current terminal print channel, otherwise 0.

NLS SYSTEM Function Values

Value	Token	Return Value
105	NLS\$CONFIG	A dynamic array, with the elements separated by field marks, containing the current values of the parameters in the <i>uvconfig</i> file related to NLS maps; otherwise 0. See the UVNLS.H include file for a list of equate tokens that define the order of the fields.
106	NLS\$SEQMAP	The current name of the map used for sequential I/O, otherwise 0. This is the value for the NLSDEFSEQMAP parameter unless it is overridden by a SET.SEQ.MAP command.
107	NLS\$GCIMAP	The name of the current GCI map.

NLS SYSTEM Function Values (Continued)

The UVNLS.H include file also gives the internal character set values of the UniVerse system delimiters.

Here is a program example that examines the current NLS settings:

```

$INCLUDE UNIVERSE.INCLUDE UVNLS.H
IF SYSTEM(NLS$ON)
THEN PRINT "Terminal map set to: ":SYSTEM(NLS$TERMMAP)
ELSE PRINT "NLS is not enabled"

```

String Length

UniVerse BASIC uses characters rather than bytes to determine string length. Statements and functions such as LEN, MATCH, INDEX, FIELD, TRIM, REPLACE, READ, WRITE, PRINT, and so on, work in the same way for multibyte and single-byte character sets.

Statements and functions that operate on dynamic arrays, for example, EXTRACT, REMOVE, INSERT, DELETE, and so on, work equally well with NLS turned on or off. This is because they look for UniVerse system delimiters in string variables, which have the same value whether NLS is on or off.

Length of Record IDs

Record IDs in UniVerse files must not exceed 255 *bytes*. This means that the maximum number of *characters* in a record ID depends on the character set in use. For multibyte character sets, the safe limit is 85 characters. This allows each character to be three bytes long in the internal character set.

This limit also applies to values used as keys in secondary indexes. If a secondary index is too long, a WRITE statement fails, a message is issued, and a nonzero value is returned to the STATUS function.

Display Length in BASIC

UniVerse BASIC uses character maps to find the correct display length for a character. Several UniVerse BASIC statements and functions can operate on the display length rather than the character length.

- The [LENDP function](#) and [LENSDP function](#) distinguish display length from character length.
- The [HEADING statement](#) and the [FOOTING statement](#) allow for varying display positions in gaps.
- The [FMTDP function](#), [FMTSDP function](#), and the [FOLDDP function](#) work like the [FMT function](#), [FMTS function](#), and the [FOLD function](#), but use display positions rather than character lengths.
- The [SETPTR statement](#) allows you to associate a map with a print channel. This means you can determine display widths for formatting spooled output. (Note that the internal to external mapping does not take place until a report is printed.)
- The [INPUTDP statement](#) works like the [INPUT statement](#), but allows you to define input displays to work in terms of variable display positions.

The display length of the unknown character is assumed to be 1.

For the syntax and full details about these statements and functions, see *UniVerse BASIC*.

Finding the Display Length of a String

Use the [LENDP function](#) and the [LENSDP function](#) to return the display length of a string. These functions are similar to the [LEN function](#) and the [LENS function](#), respectively. If these functions are executed with NLS turned off, the program behaves as if the equivalent LEN or LENS function had been called.

Formatting a String in Display Positions

Use the [FMTDP function](#) and the [FMTSDP function](#) to format a string in display positions rather than character lengths. If these functions are executed when NLS is not enabled, the program behaves as if the [FMT function](#) or the [FMTS function](#) had been called.

Folding Strings Using Display Positions

Use the [FOLDDP function](#) to fold a string using the display position length rather than its length in characters. If FOLDDP is executed when NLS is not enabled, the program behaves as if the [FOLD function](#) had been called.

Inputting Using Display Length with INPUTDP

The [INPUTDP statement](#) is equivalent to the [INPUT statement](#), but it works on character display lengths.

Inputting Through a Mask with INPUT @

Display positions affect how masks work with an `INPUT @` statement. If the external character set is multibyte, the initial value is displayed through the mask as far as possible. If you enter a new value, the mask disappears, and the user inputs to a field of the appropriate length not including any inserted characters.

The only editing functions supported are backspace and kill. When the user finishes inputting, the new value is redisplayed through the mask just as the original value was.

Block Size Always in Bytes

With the [READBLK statement](#) and the [WRITEBLK statement](#), you must specify the block size in bytes, not characters. This is because these statements are normally used to read binary data in blocks. However, the data read is mapped using the appropriate file map, so the strings that are read can be processed in the internal character set using any BASIC functions.

Similarly, you must be careful about block sizes for tapes written in a multibyte external character set. Data is written in blocks of *bytes*, and if you specify an odd number, you may get a character split across a block boundary. In particular, the [READT statement](#) may return a status value indicating that an unmappable character was read, and the [WRITET statement](#) will truncate a string, possibly writing an incomplete character.

The REMOVE Pointer and Multibyte Character Sets

When you use the [SETREM statement](#) to set the REMOVE pointer of a dynamic array, the position you specify for the REMOVE pointer must be calculated in *bytes*, not characters. You should not call SETREM and give it a random integer, since it may not point to the start of a character in the internal character set. You should use only a value returned by GETREM, which is guaranteed to be correct.

Maps in UniVerse BASIC

UniVerse BASIC statements that perform input or output always map external data to the UniVerse internal character set using the appropriate map for the device or file. In addition to the statements previously discussed, the following statements also use maps for input and output:

Device	Statements
Terminals	CRT, INPUT, INPUTIF, PRINT, and TPRINT
Printers	PRINT with PRINTER ON
Files	MATREAD, MATREADL, MATREADU, MATWRITE, MATWRITEU, READ, READL, READT, READU, READV, READVL, READVU, READSEQ, WRITESEQ, WRITE, WRITET, WRITEU, WRITEV, and WRITEVU
Tapes	READT and WRITET
Sequential files, etc.	OPENDEV, OPENSEQ, READSEQ, and WRITESEQ

Statements Using Maps

Determining a File's Map Name

In NLS mode, each UniVerse file has an associated map that defines the external character set for the file. If your program opens and reads a file, you may need to know the name of the map associated with the file to ensure that the file map is the one that your program expects. There are two main ways you can use to determine the map name:

- Calling the [FILEINFO function](#)
- Executing a GET.FILE.MAP command

The ANALYZE.FILE and FILE.STAT commands also include the map name for the file in their reports.

FILEINFO Function

To use the FILEINFO function to determine a file's map name, use the FINFO\$NLSMAP value. A token is defined in the FILEINFO.H include file as follows:

Value	Token	Returns...
20	FINFO\$NLSMAP	The file map name if NLS is enabled, or an empty string. If the file's map is a default specified in the <i>uvconfig</i> file, the returned string is the map name followed by the name of the configurable parameter in parentheses.

The following example returns the map currently used by the VOC file:

```
$INCLUDE UNIVERSE.INCLUDE FILEINFO.H
OPEN "VOC" TO filevar
ELSE STOP "Cannot open the VOC file"
mapname = FILEINFO(filevar, FINFO$NLSMAP)
PRINT "Map in use for the VOC is: ":FIELD(mapname, '(', 1)
```

Maps for Source Files

If you use embedded literal strings containing non-ASCII characters, you must specify a map for the source code in one of the following ways:

- Ensure that the source file has a map defined for it. If the file itself has no explicit map, you can specify the default map to use in the NLSDEFDIRMAP configurable parameter in the *uvconfig* file.
- Specify the \$MAP *mapname* compiler directive. The map must be installed in UniVerse, or the compiler produces an error. Only one \$MAP directive line is allowed during the compilation; multiple lines cause a compilation error. For more information, see *UniVerse BASIC*.

Note: *Programs containing non-ASCII characters that were compiled in NLS mode cannot be run with NLS mode off. Programs that contain ASCII characters can always be run, whether NLS mode is on or off.*



Maps and Devices

This section gives more information about how maps are used by devices. For information about configuring devices, see [Associating Maps with Devices](#) in Chapter 2, “Getting Started.”

Maps for Auxiliary Devices

If there is an auxiliary device associated with a terminal, a program can send data to the device in the correct character set. It does this by using an auxiliary map defined through the AUXMAP statement. This avoids having to hard code the map name.

@ Function Codes for Terminal and Auxiliary Maps

There are two *terminfo* records that you can use to set maps for terminals and auxiliary printers as follows:

Integer	Equate Name	Description
-80	IT\$NLSMAP	Main terminal map name
-81	IT\$NLSAUXMAP	Auxiliary printer map name

terminfo Records

If these map entries are not set in the *terminfo* file, the default specified in the NLSDEFTERMMAP parameter of the *uvconfig* file is used. If the *terminfo* record specifies maps that are not installed, the defaults are used and you may see a warning.



Warning: *The maps named in terminfo may not be the current terminal map. For example, the value can be overridden by a SET.TERM.TYPE command. Do not use the TERMINFO function or the @ function to read the terminfo values. Use the !GETPU subroutine, the GET.TERM.TYPE command, or the SYSTEM function instead.*

Printing Previously Mapped Data with UPRINT

You can use the UPRINT statement to print data that has already been mapped to an external format using OCONV NLS*mapname* (see “[NLS Conversion Code](#)” on page 22). The data is not mapped again by the printer’s map. If NLS is not enabled, UPRINT behaves like PRINT.

Finding the Map Associated with a Print Channel

You can use the !GETPU subroutine to determine the map name associated with a print channel using the following token, which is defined in the GETPU.H include file:

Value	Token	Returns...
22	PU\$NLSMAP	The print channel’s map name if NLS is enabled, or an empty string.

If this token is used to call !GETPU when NLS is disabled, the following run-time warning message is issued:

```
Program "!GETPU": pc = nnnn, Unsupported option "PU$NLSMAP".  
Ignored.
```

This code example finds the name of the map associated with print channel 0:

```
$INCLUDE UNIVERSE.INCLUDE GETPU.H  
CALL !GETPU(PU$NLSMAP, 0, mapname, code)  
PRINT "Map in use for print unit 0 is: ":mapname
```

Maps for UNIX Pipes

You can assign maps to UNIX pipes opened with the [OPENDEV statement](#) or the [OPENSEQ statement](#). OPENDEV assigns maps to devices and OPENSEQ assigns maps to sequential files and pipes.

OPENDEV uses the map name in the entry in the &DEVICE& file to open a UNIX device. The NLSDEFDEVMAP parameter contains the default map name. Use the ASSIGN command to override the NLSDEFDEVMAP parameter.

OPENSEQ *filename, record.id* uses the map assigned to the type 1 or type 19 file in the *.uvnlsmmap* file. If there is no map name, the map name in the NLSDEFDIRMAP parameter is the default. Use the SET.FILE.MAP command to override the NLSDEFDIRMAP parameter.

OPENSEQ *pathname* opens a UNIX pipe, file, or special device directly. OPENSEQ uses the map name in the directory containing *pathname*. If there is no map name, the map name in the NLSDEFSEQMAP parameter is the default. Use the SET.SEQ.MAP command to override the NLSDEFSEQMAP.

The SET.SEQ.MAP command specifies the map to use with BASIC sequential I/O statements if you cannot find an explicit map in the sequential file that you opened. For details about [SET.SEQ.MAP](#), see the *UniVerse User Reference*.

Unmappable Characters

A character that cannot be mapped using the current map is called an unmappable character. If UniVerse encounters unmappable characters during a read or write, its behavior is determined by two factors:

- The setting of the NLSREADELSE and NLSWRITEELSE parameters in the *uvconfig* file
- Whether there is an ON ERROR clause

The STATUS function returns values to indicate the treatment of the unmappable characters, as described in the next sections.

Unmappable Characters and WRITE Statements

If UniVerse encounters unmappable characters while executing WRITE statements, that is, WRITE, WRITEU, WRITEV, WRITEVU, and MATWRITE, the STATUS function returns certain values. The values returned and the behavior of UniVerse depend on the existence of an ON ERROR clause and the setting of the NLSWRITEELSE parameter.

The STATUS function returns certain values when an ON ERROR clause is present and the NLSWRITEELSE parameter is set to 1. The write fails and no records are written.

- If the unmappable character is in the record ID, the STATUS function returns 3.
- If the unmappable character is in the record's data, the STATUS function returns 4.

The behavior of UniVerse is different when there is no ON ERROR clause and the NLSWRITEELSE parameter is set to 1. The following occurs:

- If the unmappable character is in the record ID, the program aborts with a message in this format:

```
Program "name": Line nnn,  
Record Id ? contains characters which are not defined in the  
file's NLS map.
```

- If the unmappable character is in the record's data, the program aborts with a message in this format:

```
Program "name": Line nnn,  
Record record.id contains characters which are not defined in the  
file's NLS map.
```

The behavior of UniVerse also varies when there is no ON ERROR clause and the NLSWRITEELSE parameter is set to 0. The following occurs:

- If the unmappable character is in the record ID, the program aborts with a message in this format:

```
Program "name": Line nnn,  
Record Id ? contains characters which are not defined in the  
file's NLS map.
```

Regardless of the existence of an ON ERROR clause, if NLSWRITEELSE is set to 0 and the unmappable character is in the record's data, UniVerse writes the record using the map's unknown character to replace the unmappable characters. The unknown character is usually a question mark (?). Data is lost as a result.

Note: There is no relationship between the NLSWRITEELSE parameter and the ELSE clause of a UniVerse BASIC statement.



Unmappable Characters and READ Statements

If UniVerse encounters unmappable characters during READ statements, that is, READ, READU, READV, READVU, and MATREAD, the STATUS function returns certain values. The values returned and the behavior of UniVerse depend on the existence of the setting of the NLSREADELSE parameter.

The STATUS function returns certain values when the NLSREADELSE parameter is set to 1. Depending on the origin of the unmappable characters, the following occurs:

- If the unmappable character is in the record ID, the program takes the ELSE clause and the STATUS function returns 3 with a message in this format:

```
Program "name": Line nnn,  
Record Id ? contains characters which are not defined in the  
file's NLS map.
```


- If the unmappable character is in the record's data, the program takes the ELSE clause and the STATUS function returns 4. You also see a message in this format:

```
Program "name": Line nnn,  
Record record.id contains characters which are not defined in the  
file's NLS map.
```

The behavior of UniVerse differs when the NLSREADELSE parameter is set to 0. Depending on the origin of the unmappable characters, the following occurs:

- If the unmappable character is in the record ID, the program takes the ELSE clause and the STATUS function returns 3.

Note: This is different from the case when a record does not exist, where STATUS returns 0.

- If the unmappable character is in the record's data, the record is read, and the unmappable characters are replaced with the Unicode replacement character (value xFFFD). No message is displayed, and data is lost.



ASCII and EBCDIC Conversions

The ASCII and EBCDIC functions convert between 7-bit ASCII values and 8-bit EBCDIC values. The functions work the same way whether NLS mode is on or off. This may result in ambiguous data that is not recognized by your current mapping, for example, terminal maps, file maps, and so forth.

Multinational Characters in UniVerse BASIC

All UniVerse BASIC language elements in source code, such as paths, variable names, tokens, subroutine names, and reserved words, must be in 7-bit US ASCII. You can use other character sets in your source code for the following:

- Embedded literal strings. In this case there must be a map associated with the source file. For more information about maps, see Chapter 3, “Maps.”
- Comments.

You can specify any Unicode value using the UNICHAR function. See “[CHAR and SEQ in NLS Mode](#)” on page 21. You can specify certain 8-bit characters in your source by using CHAR (nnn), where *nnn* is a decimal value 129 through 247.

Note: If your program source uses a CHAR (nnn) function, it must be recompiled for use in NLS mode.



Editing Multinational Characters

You can use ED to edit multinational characters in records and source code. With NLS mode enabled, ED offers a further up-arrow mode to deal with the full internal character set. Up-arrow mode can be in three states:

- Disabled
- Enabled
- Enabled+Unicode

The command ^ toggles between enabled or disabled. With NLS enabled, the command ^X switches to Unicode mode (enabled+Unicode).

In disabled mode, all characters are printed directly; whether you see them or not depends on your terminal and terminal map.

In enabled mode, code points less than 248, and system delimiters (code points 248 through 255), print using the decimal notation ^*ddd*. Every other code point uses the hexadecimal notation ^*xhhhh*, which can be entered in Unicode mode.

In enabled+Unicode mode, code points 128 (character string used to represent the null value) and 248 through 255 print in decimal notation ^*ddd*; all other code points greater than 126 use the hexadecimal notation ^*xhhhh*.

The special cases of ^094, ^128, and ^248–^255 appear in decimal in both of the enabled modes.

Note the distinction between, for example, the character printed as ^253 and that printed as ^x00FC. The first is a UniVerse value mark, the second is the lowercase y acute character.

The following tables compare the differences between inputting and displaying characters in hexadecimal and decimal notation in the two up-arrow modes:

Mode	Printed	Input in ^ddd	Input in ^xhhh
enabled	000–126	127–255	0x0100–0xFFFF
enabled+Unicode	000–126	128, 248–255	0x007F, 0x0081–0xFFFF

Hexadecimal and Decimal Notation

Special Characters (Unicode Format)	Input Format (Enabled)	Input Format (Enabled + Unicode)
CIRCUMFLEX ACCENT	^094	^094
The null value	^128	^128
C1 control character (PAD)		^x0080
UniVerse reserved mark	^248	^248
UniVerse reserved mark	^249	^249
UniVerse reserved mark	^250	^250
UniVerse text mark	^251	^251
UniVerse subvalue mark	^252	^252
UniVerse value mark	^253	^253
UniVerse field mark	^254	^254
UniVerse item mark	^255	^255

Special Characters

Special Characters (Unicode Format)	Input Format (Enabled)	Input Format (Enabled + Unicode)
LATIN SMALL LETTER O WITH STROKE	^x00F8	^x00F8 (ø)
LATIN SMALL LETTER U WITH GRAVE	^x00F9	^x00F9 (ù)
LATIN SMALL LETTER U WITH ACUTE	^x00FA	^x00FA (ú)
LATIN SMALL LETTER U WITH CIRCUMFLEX	^x00FB	^x00FB (û)
LATIN SMALL LETTER U WITH DIAERESIS	^x00FC	^x00FC (ü)
LATIN SMALL LETTER Y WITH ACUTE	^x00FD	^x00FD
LATIN SMALL LETTER THORN	^x00FE	^x00FE
LATIN SMALL LETTER Y WITH DIAERESIS	^x00FF	^x00FF (ÿ)

Special Characters (Continued)

Inputting Unicode Characters

To enter a character by its Unicode value, you can type either `^ddd` or `^xhhhh`, where *hhhh* must be a 4-digit hexadecimal number. You can use `^ddd` only for values 0 through 255.

You can input system delimiters only by using the decimal notation `^ddd`.

Generating Characters in External Format

You can use the UNICHAR function to generate a single character from a supplied Unicode value, or you can use the UNICHARS function to generate a dynamic array of characters. The UNICHAR and UNICHARS functions operate in the same way whether NLS mode is on or off.

Generating System Delimiters and the Null Value

Do not use UNICHAR or UNICHARS to generate UniVerse system delimiters or the internal representation of the null value. Use the BASIC @variables instead: @TM, @SVM, @SM, @VM, @FM, @AM, @IM, and @NULL.STR.

Generating Characters in Internal Format

You can generate a Unicode value from a supplied character using the UNISEQ function, or you can generate a dynamic array of Unicode values using the UNISEQS function. These functions perform the opposite action of the UNICHAR and UNICHARS functions.

CHAR and SEQ in NLS Mode

Use the CHAR and SEQ functions with care in NLS mode.

Use CHAR (*nnn*) to operate modulo 256. If *nnn* is in the range 0 through 127, 128, and 248 through 255, it operates in the same way whether NLS mode is on or off. If *nnn* is in the range 129 through 247, it produces Unicode characters in the range x0081 through x00F7. These correspond to the ISO 8859-1 (Latin 1) characters with those values, and are multibyte characters. If you want to generate the specific *bytes* with those values, use the BYTE function. To generate characters outside the CHAR range, use UNICHAR. For more information, see *UniVerse BASIC*.

Use SEQ (*var*) to return a number in the range 0 through 255, but you cannot use this function to look at the Unicode values in the range x0080, and x00F8 through x00FF, or above. To examine those values, use UNISEQ. If you call SEQ on a character outside its range, a run-time message is printed, and an empty string is returned.

Internal and External String Conversion

You can use the ICONV and OCONV functions to do the following:

- Convert an internal Unicode string to its external representation and vice versa, using the NLS conversion code
- View internal strings in their Unicode hexadecimal format using the MU0C conversion code

NLS Conversion Code

Use the following syntax for ICONV and OCONV with the NLS conversion code:

ICONV (*string*, "NLSmapname")

OCONV (*string*, "NLSmapname")

ICONV treats *string* as being in the external format defined by *mapname*, converts it to internal format, and returns the result. Use OCONV to convert *string* from internal format to the external format specified by *mapname*.

mapname must be either the name of an installed map or one of the special strings LPTR, CRT, AUX, or OS. These denote the map associated with the current printer, terminal, auxiliary printer, or operating system respectively. With ICONV, if *mapname* is the value UNICODE, each two bytes in *string* is assumed to be a Unicode character. If there is an odd number of bytes in *string*, Universe substitutes the last byte with the Unicode replacement character (xFFFD) and the STATUS function returns 3. If *mapname* is not installed, an empty string is returned.

The conversion works only with NLS mode on. The STATUS function can return the following values:

Value	Description
0	The conversion succeeds.
1	The map name supplied is invalid, an empty string is returned.
2	The conversion is invalid or NLS is not enabled.
3	Some characters of the converted string could not be mapped, and the returned string contains replacement characters.

STATUS Function Return Values

Use UPRINT instead of PRINT (which treats string as being in internal format) to print the external format string returned by OCONV NLS*mapname*.

For example:

```
UPRINT OCONV(VAR, "NLSSHIFT-JIS")
```

For more information, see *UniVerse BASIC*.

MU0C Conversion Code

Use the MU0C conversion code to view internal strings in Unicode hexadecimal format.

Note: The MU0C conversion code uses four hexadecimal digits. The MX0C conversion code treats strings as two hexadecimal digits per byte, and does not know about internal Unicode format.



Use the following syntax:

```
ICONV (string, "MU0C")
```

```
OCONV (string, "MU0C")
```

If you use the conversion code with the UniVerse system delimiters, note that OCONV(@FM, "MU0C") returns xF8FE, and ICONV("F8FE", "MU0C") produces @FM, that is, the single character CHAR(254) in internal format. This is so you can distinguish UNICHAR(254) from CHAR(254). OCONV(UNICHAR(254), "MU0C") returns x00FE.

The value of the BASIC STATUS function after an MU0C conversion has been executed is as follows:

Value	Description
0	The conversion succeeds.
2	The conversion is invalid or NLS is not enabled.

BASIC STATUS Function Return Values

The following example shows internal to external byte sequences for several characters:

```
X = UNICHAR(222):UNICHAR(240):@FM
PRINT "Internal form in hex bytes is: ":OCONV(X, 'MX0C')
Y = OCONV(X, 'NLSISO8859-1')
PRINT "External form in hex bytes is: ":OCONV(Y, 'MX0C')
PRINT "Internal form in Unicode is: ":OCONV(X, 'MU0C')
```

This program produces the following output:

```
Internal form in hex bytes is: C39E   C3B0   FE
External form in hex bytes is:  DE    F0    3F
Internal form in Unicode is:   00DE   00F0   F8FE
```

The characters in the output are separated by spaces in order to display the differences more easily. For example, C39E represents 222 in the internal form in UniVerse, DE represents 222 in the external byte sequence as it is displayed on the terminal, and 00DE represents 222 in the Unicode byte sequence.

Likewise, C3B0 represents 240 in the internal form in UniVerse, F0 represents 240 in the external byte sequence for the terminal, and 00F0 represents 240 in the Unicode byte sequence.

In the final column, FE is the internal representation of @FM, 3F (the Unicode character ?) represents the external byte sequence for the terminal, and F8FE represents the Unicode byte sequence.

Other Conversion Codes

You can use other conversion codes with ICONV and OCONV, such as MM (monetary conversion), NL (Arabic numeral conversion), MCM, MC/M, and MCW (additional masked character conversions). For more information about these conversion codes, see *UniVerse BASIC*.

Displaying Records by Character Value

You can check the contents of a record even if your terminal cannot display the character set that the record uses.



Warning: *Be careful to distinguish the differences in how characters are represented on your terminal. A system delimiter, for example @VM, is displayed as FC in the HEX case, but F8FC in the UNICODE case, not 00FC. F8FC is the external representation of the UniVerse value mark in Unicode. The value remains unchanged.*

The **COPY**, **CP**, and **CT** commands have a **HEX** option to display the contents of a record in hexadecimal digits, and a **UNICODE** option to display the Unicode values of the characters. For the Pick version of the **COPY** verb, you specify (U instead of **UNICODE**, and (H instead of **HEX**.

For example, if a record contains the string ABC in field 1 and ÄBÇ in field 2, using the **HEX** option, you see the following with NLS mode off. In field 1 the 41 is the ASCII code for A, and C4 is the (single byte) ASCII code for Ä.

```
>COPY FROM VOC 'EXAMPLE' CRT HEX
```

```
EXAMPLE  
0001 414243  
0002 C4DFC7
```

You see the following with NLS mode on:

```
>COPY FROM VOC 'EXAMPLE' CRT HEX
```

```
EXAMPLE  
0001 414243  
0002 C384C39FC387
```


ABC uses one byte per character in internal format (line 0001) whereas ÄBÇ uses two bytes per character (line 0002). Field 1 contains 41, the (single byte) internal code for A, and field 2 contains C384, the (double byte) internal code for Ä.

Using the UNICODE option you see the following:

```
>COPY FROM VOC 'EXAMPLE' CRT UNICODE  
  
EXAMPLE  
0001 004100420043  
0002 00C400DF00C7
```

Line 0001 is zero-extended, but similar to the previous example, whereas line 0002 is completely different. 0041 is the UNICODE representation for A, and 00C4 is the UNICODE for Ä.

Exchanging Character Values

The UniVerse BASIC EXCHANGE function is not NLS-aware and may not produce the results you expect when NLS is enabled. This function has two arguments: the first is the hexadecimal value of a character to be found, and the second is a hexadecimal value of a character to replace it with. EXCHANGE looks at only the first two bytes of its arguments and so can handle only characters 00 through FF. In NLS mode, bytes 00 through FA are treated as Unicode characters 0000 through 00FA, and bytes FB through FE are treated as system delimiters. If FF is used as the second argument, all occurrences of the character designated by the first argument are deleted.

Case Inversion and Deadkey Characters

Deadkey characters are generated by a sequence of keystrokes rather than a single, dedicated key. Deadkey characters are always generated *after* any case inversion commands are processed. This means that a command such as PTERM CASE INVERT has no effect on characters entered through deadkey sequences.

For example, using the MNEMONICS map, if case inversion is on (the default), entering the sequence < a- > produces the character LATIN SMALL LETTER A WITH MACRON (not < A- >, the character LATIN CAPITAL LETTER A WITH MACRON).

BASIC and Locales

A locale comprises the set of conventions in the five categories (Time, Numeric, Monetary, Ctype, and Collate). From within a UniVerse BASIC program, you can do the following:

- Retrieve the current locale names in any category
- Save the current locale setting
- Restore the saved locale setting
- Change the current locale setting

For information about setting locales systemwide, see Chapter 4, “[Locales](#).”

Retrieving Locale Settings

You can retrieve locale settings using the [GETLOCALE function](#) and the [LOCALEINFO function](#). GETLOCALE retrieves the names of specified categories of the current locale. LOCALEINFO retrieves the settings of the current locale.

Saving and Restoring Locales

You can save and restore locales using the [SETLOCALE function](#) with the UVLC\$SAVE and UVLC\$RESTORE tokens.

Changing the Current Locale

You can change or disable a locale setting using the SETLOCALE function.

NLS in Client Programs

Client Programs	6-3
Maps	6-3
Locales	6-4
System Delimiters and the Null Value	6-4
UniObjects.	6-5
UniObjects for Java and UniObjects for .NET	6-6
InterCall Functions	6-7
UCI Programs.	6-8
Connecting to the Server	6-8
Requesting an SQLConnect	6-8
Setting the Map and Locale	6-8
Values in the UCI Configuration File	6-9
Interpreting the Map Name	6-10
Interpreting the Locale Name	6-10
Using SQLGetInfo	6-10
BCI Programs.	6-11
Connecting to the Server	6-11
Requesting an SQLConnect	6-11
Setting the Locale	6-11
Interpreting the Locale Name	6-12
Using SQLGetInfo	6-13
GCI Subroutines	6-14
Specifying Maps for GCI Subroutines.	6-14
Data Types for Multibyte Characters	6-14

This chapter describes how client programs and external subroutines use NLS. The topics covered include:

- Points to watch when you write client programs
- How to access NLS functionality from the following APIs:
 - GCI (General Calling Interface)
 - BCI (UniVerse BASIC SQL Client Interface)
 - UniVerse ODBC
 - UCI
 - InterCall
 - UniObjects
 - UniObjects for Java
 - UniObjects for .NET

These APIs (except for GCI) access client/server technology.

Client Programs

This section contains some general information about using NLS features from client programs. A server process is a UniVerse instance that processes requests from a client process and produces results. A server platform refers to the computer on which the server processes run.

The most important point to remember is that you cannot access any NLS functionality unless NLS mode is enabled on the server. All servers honor the settings of the UniVerse configurable parameters and client requests for character mapping and locales.

For all client/server programs, the UniRPC must be running on the server platform.

You need to be aware of the following relevant configurable parameters:

Parameter	Description
NLSMODE	Enables NLS.
NLSDEF SRV MAP	Name of the default map to be used when passing string arguments to or from client. This is used if the client does not specify a map.
NLSLCMODE	Enables locale mode if NLS is enabled.
NLSDEF SRV LC	Name of the default locale to be used when communicating with client. This is used if the client does not specify a locale.

Configurable Parameters

Maps

Except for BCI clients, UniVerse performs character mapping on the server. Your program can inform the server of the appropriate map name or the character set you use to send and receive data. In theory, you can set and reset maps as many times as you want in a program. All users who log on to a client/server system have their own individual copies of the server program.

Since BCI client programs run as part of UniVerse, the client performs its own character mapping.

Locales

UniVerse does locale conversions on the server. Your program must inform the server of the locale your programs want to use in order to send or receive data. Once a connection to the server is established, you can change the locale settings as you wish. These settings do not interfere with other client/server users.

System Delimiters and the Null Value

Your program must use the correct values for system delimiters and the null value. You should not use hard-coded values for system delimiters. Always interrogate a UniVerse variable (for example, @VM), or use the equivalent functionality in your API, for example, the VM property of the Session object in UniObjects, or SQLGetInfo for UCI.

UniObjects

UniObjects provides two objects that use NLS. The NLSMap and NLSLocale objects are accessed through the Session object, which has several properties associated with NLS. If NLS mode is not enabled on the server, these objects are not available and return an empty string. For more information, see the *UniObjects Developer's Guide*.

NLSLocale Object

Use the NLSLocale object to define and manage the locale setting for the Time, Numeric, Monetary, Ctype, and Collate categories. You can supply the values for these categories as a single DynamicArray object, with five elements. This object has no default property.

UniObjects for Java and UniObjects for .NET

UniObjects for Java and UniObjects for .NET provide two objects that use NLS. The UniNLSMap and UniNLSLocale objects are accessed through the UniSession object, which has several methods associated with NLS. If NLS mode is not enabled on the server, these objects are not available and throw an exception. For more information, see the *UniObjects for Java Developer's Guide* or the *UniObjects for .NET Developer's Guide*.

UniNLSMap Object

The UniNLSMap object defines the map to be used on the server for the UniObjects for Java session.

UniNLSLocale Object

Use the UniNLSLocale object to define and manage the locale setting for the Time, Numeric, Monetary, CType, and Collate categories. You can supply the values for these categories as a single UniDynArray object with five elements. This object has no default method.

InterCall Functions

The following InterCall functions use NLS functionality.

- `ic_get_locale`
- `ic_get_map`
- `ic_get_mark_value`
- `ic_set_locale`
- `ic_set_map`

For more information about InterCall, see the *InterCall Developer's Guide*.

UCI Programs

UCI programs get their default settings for maps and locales from the operating system (on Windows platforms) and from the UCI configuration file.

UCI programs communicate with a UniVerse server running on a platform. If NLS is enabled on the platform, the data returned to the UCI program reflects the NLS character mapping and locale settings. The UCI program can also examine and control these settings.

UCI programs need to be aware of the way NLS affects the retrieval and sending of data. For more information about UCI, see the *UCI Developer's Guide*.

Connecting to the Server

When a UCI program tries to connect to a UniVerse server, a server process begins to process the requests from the UCI program.

When the server starts, it behaves like UniVerse on that platform. For example, it uses the configurable parameters that are initialized in the *uvconfig* file. The UniRPC must be running on the server platform.

Requesting an SQLConnect

When the UCI program requests an SQLConnect, it performs the following actions specific to NLS:

- The server tells the client which UniVerse version is running.
- If the release is UniVerse 9.4 or later, the client automatically tries to specify the NLS character map.
- If it succeeds, NLS is enabled on the server, and the client then tries to set the locale.

Setting the Map and Locale

The client can check the following places to determine the map and locale to set:

- The operating system for map and locale names.

- The UCI configuration file for the values that it used to locate the UniVerse server. The values in the UCI configuration file override the operating system's map and locale settings.

Like the other UCI configuration file values, you can specify these values once for all UniVerse data sources, or individually for each data source.

Values in the UCI Configuration File

The following table describes the values in the UCI Configuration file.

Value	Description
NLSMAP	Name of the map to use when passing string arguments to/from client routine. The map name must be 7-bit ASCII.
NLSLOCALE	Name of the locale to use when communicating with client routines. Uses characters in NLSMAP. Specifies values for the following categories:
NLSLCTIME	Name of the locale whose TIME category is to be used.
NLSLCNUMERIC	Name of the locale whose NUMERIC category is to be used.
NLSLCMONETARY	Name of the locale whose MONETARY category is to be used.
NLSLCCTYPE	Name of the locale whose CTYPE category is to be used.
NLSLCCOLLATE	Name of the locale whose COLLATE category is to be used.
NLSLCALL	Same as NLSLOCALE if one locale is specified, or can specify five locale names separated by the slash character (/).

UCI Configuration File Values

The UCI program can override these values with the corresponding SQLSetConnectOption calls. For example:

```
status=SQLSetConnectOption(hstmt, SQL_NLSMAP, 0, "WIN:850")
```

The UCI program can use similar calls to change the NLS behavior of a connection any time a transaction is not active.

Interpreting the Map Name

The server interprets the map name as follows:

- If the map name is in NLS.CLIENT.MAPS, use the corresponding value.
- If the map is installed, use it.
- Replace ':xxx' at the end of the map name with ':DEFAULT' and look in NLS.CLIENT.MAPS; if found, use the corresponding value. Otherwise the map is unchanged and an error is returned.

Interpreting the Locale Name

The server interprets the locale name as follows:

- If the locale name is in NLS.CLIENT.LCS, use the corresponding value.
- If the locale is installed, use it.
- Replace ':xxx' at the end of the locale name with ':DEFAULT' and look in NLS.CLIENT.LCS; if found, use the corresponding value. Otherwise, the locale is unchanged and an error is returned.

Using SQLGetInfo

A UCI program can use the SQLGetInfo function to determine the current NLS state of the server. Also, to read or write data that includes UniVerse system delimiters or the null value, the program must determine the characters to which they are mapped. SQLGetInfo can provide this information. The previous SQL values are defined in *UCI.h*.

BCI Programs

BCI programs use UniVerse NLS only when they connect to UniVerse servers. Since BCI programs run as part of UniVerse, the client performs its own character mapping.

BCI programs get their default settings for locales from the operating system, the UniVerse environment, and the *uvodbc.config* file.

BCI programs communicate with a UniVerse server running on a platform. If NLS is enabled on the platform, the data returned to the BCI program reflects the NLS locale settings. The BCI program can also examine and control these settings.

BCI programs need to be aware of the way NLS affects the retrieval and sending of data. For more information about BCI, see the *UniVerse BASIC SQL Client Interface Guide*.

Connecting to the Server

When a BCI program tries to connect to a UniVerse server, a server process begins to process requests from the BCI program.

When the server starts, it behaves like UniVerse on that platform. For example, it uses the configurable parameters that are initialized in the *uvconfig* file. The UniRPC must be running on the server platform.

Requesting an SQLConnect

When the BCI program requests an SQLConnect, it performs the following actions specific to NLS:

- The server tells the client which UniVerse version is running.
- If it succeeds, NLS is enabled on the server, and the client then tries to set the locale.

Setting the Locale

The client can check the following places to determine the locale to set:

- The operating system for locale names.

- The current UniVerse environment for locale names.
- The *uvodbc.config* file for the values that it used to locate the UniVerse server. The values in the *uvodbc.config* file override the locale settings of the operating system and the UniVerse environment.

Like the other *uvodbc.config* values, these values can be specified once for all UniVerse data sources, or individually for each data source.

Values in the *uvodbc.config* File

Value	Description
NLSLOCALE	Name of the locale to use when communicating with client routines. Specifies values for the following categories:
NLSLCTIME	Name of the locale whose TIME category is to be used.
NLSLCNUMERIC	Name of the locale whose NUMERIC category is to be used.
NSLSCMONETARY	Name of the locale whose MONETARY category is to be used.
NLSLCCTYPE	Name of the locale whose CTYPE category is to be used.
NLSLCCOLLATE	Name of the locale whose COLLATE category is to be used.
NLSLCALL	Same as NLSLOCALE if one locale is specified, or can specify five locale names separated by the slash character (/).

uvodbc.config File

The BCI program can override these values with the corresponding SQLSetConnectOption calls. For example:

```
status=SQLSetConnectOption(hdbc, SQL.UVNLS.LOCALE, CA-FRENCH)
```

The BCI program can use similar calls to change the NLS behavior of a connection any time a transaction is not active.

Interpreting the Locale Name

The server interprets the locale name as follows:

- If the locale name is in NLS.CLIENT.LCS, use the corresponding value.
- If the locale is installed, use it.

- Replace ':xxx' at the end of the locale name with ':DEFAULT' and look in NLS.CLIENT.LCS; if found, use the corresponding value. Otherwise, the locale is unchanged and an error is returned.

Using SQLGetInfo

A BCI program can use the SQLGetInfo function to determine the current NLS state of the server. Also, to read or write data that includes UniVerse system delimiters or the null value, the program must determine the characters to which they are mapped. SQLGetInfo can provide this information. The previous SQL values are defined in *ODBC.h*.

GCI Subroutines

The two main points to note when writing GCI subroutines are as follows:

- GCI subroutines must use the correct map.
- Strings that contain multibyte characters must have the correct data type.

For complete information about GCI, see the *UniVerse GCI Guide*.

Specifying Maps for GCI Subroutines

You must ensure that GCI subroutines use the correct map for passing strings. There are two ways of doing this:

- With the NLSDEFGCIMAP parameter of the *uvconfig* file. This specifies the default map to use for all character string parameters passed to and from GCI subroutines. For more information about setting *uvconfig* parameters, see [Setting Configurable Parameters](#) in Chapter 2, “Getting Started.”
- With the SET.GCI.MAP command.

Use SET.GCI.MAP to retrieve the GCI map setting or to set a global GCI map for all input and output. The SET.GCI.MAP command overrides the default setting in the NLSDEFGCIMAP parameter. If the map name does not exist, or if NLS mode is not enabled, UniVerse returns a message. However, if you enter the command without qualifiers, UniVerse retrieves the current map setting.

Data Types for Multibyte Characters

Use the following GCI data types to specify multibyte characters:

Data Type	Description
wchar_t*	Pointer to wchar.
pwchar_t*	Pointer to preallocated string memory.

Multibyte Character Data Types

Data Type	Description
twchar_t*	Pointer to character memory that is allocated by the subroutine.
lwchar_t*	Pointer to character memory allocated by the GCI.
wchar_tvar*	Pointer to a STRING type. Contains a length and a pointer to the data.

Multibyte Character Data Types (Continued)

NLS Administration Menus

Unicode Menu 7-3

This chapter describes the structure and content of the NLS Administration menus.

You must be a UniVerse Administrator in the UV account to use the menus. To display the main NLS Administration menu, use the NLS.ADMIN command. The NLS Administration menu has the following options:

- **Unicode.** This option lets you examine the Unicode character set using various search criteria.
- **Mappings.** This option lets you view, create, or modify map descriptions or map tables.
- **Locales.** This option lets you view, create, or modify locale definitions.
- **Categories.** This option lets you view, create, or modify category files and weight tables.
- **Installation.** This option lets you install maps into shared memory or edit the *uvconfig* file.

The options lead to further menus that are described in the following sections.

Unicode Menu

Use the Unicode menu to examine the Unicode character set. The following options are available:

- **Characters.** This option leads to a further menu containing the following options:
 - **List All descriptions.** Provides a very long listing of all the Unicode characters.
 - **by Value.** Prompts you to enter a Unicode 4-digit hexadecimal value, then returns its description.
 - **by Char description.** Prompts you to enter a partial description of a character, then returns possible matches.
 - **by block Number.** Lists all characters in a given Unicode block in Unicode order.
 - **by Block descriptions.** Lists the Unicode block numbers, the official description of what each block contains, the start and end points in the Unicode set, and the number of characters in the block.
 - **Ideograph xref.** The start of further levels of menu, which are of interest to multibyte users only. These let you do the following:
 - Display a listing of how the Unicode ideographic area maps to Chinese, Japanese, and Korean standards
 - Search for a character in Unicode, given its external character set reference number
 - Convert between external encodings and standards reference numbers, for example, convert shift-JIS to row and column format
- **Alphabetic.** This option lists the NLS.CS.ALPHAS file. This file contains records that define ranges of code points within which characters are considered to be alphabetic. Use the CType category to modify these ranges.
- **Digits.** This option lists the NLS.CS.TYPES file. This file contains records that describe code points normally considered to represent the digits 0 through 9 in different scripts. Use the Numeric category to modify these ranges.

- **Non-printing.** This option lists the NLS.CS.TYPES file. This file contains records that describe code points normally considered to be nonprinting characters. Use the Ctype category to modify these ranges.
- **case Rules.** This option lists the NLS.CS.CASES file. This file describes the normal rules for converting uppercase to lowercase and lowercase to uppercase for all code points in Unicode. Use the Ctype category to modify these ranges.
- **Exit.**

Mappings Menu

Use the Mappings menu to examine, create, and edit map description and map table records, and to compile maps. The following options are available:

- **View.** Displays a listing of all map description records.
- **Descriptions.** Leads to a submenu for manipulating map descriptions, that is, records in the NLS.MAP.DESCS file. The Xref option produces a cross-reference listing that lets you see which maps and tables are being used as the basis for others.
- **Tables.** Leads to a submenu for manipulating map tables, that is, records in the NLS.MAP.TABLES file. From the submenu you can list, create, edit, delete, and cross-reference map tables.
- **Clients.** Administers the NLS.CLIENT.MAPS file, which provides synonyms between map names on a client and the UniVerse NLS maps on the server. You can list, create, edit, and delete records using this option.
- **Build.** Compiles a single map.

Locales Menu

Use the Locales menu to examine, create, and edit locale definitions. The following options are available:

- **List All.** Lists all the locales that are available in UniVerse, that is, all the records in the NLS.LC.ALL file. You may need to build the locales in order to install them into shared memory.
- **View.** Prompts you for the name of a locale, then lists the record for that locale.
- **Create.** Creates a new locale record.
- **Edit.** Edits an existing locale record.
- **Delete.** Deletes a locale record
- **Xref.** Cross-references a locale. This lets you see the relationship between various locale definitions.
- **Clients.** Administers the NLS.CLIENT.LCS file, which provides synonyms between locale names on a client, and the UniVerse NLS locales on the server. You can list, create, edit, and delete records using this option.
- **Report.** Lets you produce a report on records in locale categories. You can choose from All, Time/date, Numeric, Monetary, Ctype, and Collate.
- **Build.** Builds a locale.

Categories Menu

From the Categories menu you can administer the NLS category files for different types of convention. The following options are available:

- **Time/date**
- **Numeric**
- **Monetary**
- **Ctype**
- **Collate**
- **Weight tables**
- **Language info**

The first five options call submenus that let you list, view, create, edit, delete, and cross-reference records in the specific category. The final two options have differences as described below.

- **Weight tables.** This option has two additional suboptions as follows:
 - **Accent weights.** This option lists all the records in the NLS.WT.LOOKUP file that refer to accents.
 - **Case weights.** This option lists all the records in the NLS.WT.LOOKUP file that refer to casing.
- **Language info.** This option administers the NLS.LANG.INFO file and lets you list, view, create, edit, delete, and cross-reference records in the file.

Installation Menu

Use the Installation menu to edit the system configuration file or to install maps in shared memory. The following options are available:

- **Edit *uvconfig*.** This option lets you edit the configurable parameters in the *uvconfig* file. You can edit all the parameters, or just those referring to NLS, maps, locales, or clients.
- **Maps.** This option leads to a further menu with the following options:
 - **Configure.** Runs the NLS map configuration program.
 - **All binaries.** Lists all the built maps that are available to install into shared memory.
 - **In memory.** Lists the names of all maps currently installed in shared memory and available for use within UniVerse.
 - **(re-)Build.** Compiles a single map in the same way as the Build option on the Mappings menu.
 - **Delete binary.** Removes a binary map. This takes effect when UniVerse is restarted.
- **Locales.** This option leads to a further menu with the following options:
 - **Configure.** Runs the NLS locale configuration program.
 - **All binaries.** Lists all the built locales that are available to install into shared memory.
 - **In memory.** Lists the names of all locales currently installed in shared memory and available for use within UniVerse. Use this option if the SET.LOCALE command fails with the error `locale not loaded`. This option lets you identify locales that are built but not loaded.
 - **(re-)Build.** Compiles a single locale.
 - **Delete binary.** Removes a binary locale. This takes effect when UniVerse is restarted.
- **By language.** This option lets you configure NLS by specifying a particular language. The configuration program selects the appropriate locales and maps to be built and an appropriate configuration for the *uvconfig* file.

The NLS Database

This appendix describes the files in the NLS database. The NLS database is in the *nls* subdirectory of the UV account directory. The *nls* directory contains the subdirectories *charset*, *locales*, and *maps*.

Each subdirectory of the NLS directory contains further subdirectories, such as the *listing* and *install* subdirectories. *listing* contains listing information generated when building maps and locales (if the user selects this option). *install* contains the binary files that are loaded into memory.

You should use the NLS.ADMIN command to perform all NLS administration.

The VOC names for NLS files start with the prefix NLS (this prefix is absent if you view the files from the operating system). The second part of the file name indicates the logical group that the file belongs to. The logical groups are as follows:

These letters...	Indicate this file group...
CLIENT	Data received from client programs
CS	Information about Unicode character sets
LANG	Languages
LC	Locales
MAP	Character set maps
WT	Weight tables

Logical Groups

The third part of the file name indicates the contents of the file. For example, the file called NLS.LC.COLLATE is an NLS file belonging to the locales group that contains information about collating sequences.

The following table lists all the files in the NLS database.

File	Description
NLS.CLIENT.LCS	Defines the locales to be used by client programs connecting to UniVerse. For a description of the record format for this file, see Locales for Client Programs in Chapter 2, “Getting Started.”
NLS.CLIENT.MAPS	Defines the character set used by client programs. For a description of the record format for this file, see Maps for Client Programs in Chapter 2, “Getting Started.”
NLS.CS.ALPHAS	Defines which characters are defined as alphabetic in the Unicode standard. Each record ID is a hexadecimal code point value that indicates the start of a range of characters. The record itself specifies the last character in the range. These default values can be overridden by a national convention. You should not modify this file; it is for information only.
NLS.CS.BLOCKS	Defines the blocks of consecutive code point values for characters that are normally used together as a set for one or more languages. The record IDs are block numbers. This file is cross-referenced by the NLS.CS.DESCS file. You should not modify this file; it is for information only.
NLS.CS.CASES	Defines those characters that have an uppercase and lowercase version, and how they map between the two, according to the Unicode standard. These default values can be overridden by a national convention. Each record ID is the hexadecimal code point value for a character. You should not modify this file; it is for information only.
NLS.CS.DESCS	Contains descriptions of every character supported by UniVerse NLS. Each character has its own record, using its hexadecimal code point value as the record ID. The descriptions are based on those used by the Unicode standard. You should not modify this file; it is for information only.

NLS Database Files

File	Description
NLS.CS.TYPES	Defines which characters are numbers, nonprintable characters, and so on, according to the Unicode standard. These default values can be overridden by a national convention. Each record ID is the hexadecimal code point value for a character. You should not modify this file; it is for information only.
NLS.LANG.INFO	Contains information about languages. Provides possible mappings between language, locale and character set map. It is used for installing NLS and reporting on locales, and should not be modified.
NLS.LC.ALL	Holds records for all the locales known to UniVerse. The record IDs are the locale names. The fields of each record are the IDs of records in other locale files. These files contain data about the categories that make up a locale (Time, Numeric, and so on). For a description of the record format for this file, see Creating New Locales in Chapter 4, “Locales.”
NLS.LC.COLLATE	Each record in this file defines a collating sequence used by a locale. The collating sequences are defined according to how they differ from the default collating sequence. For a description of the record format for this file, see Format of Convention Records in Chapter 4, “Locales.”
NLS.LC.CTYPE	Each record in this file holds character typing information used in a locale, that is, which characters are alphabetic, numeric, lowercase, uppercase, nonprinting, and so on. The character types are defined according to how they differ from the default character typing. For a description of the record format for this file, see Format of Convention Records in Chapter 4, “Locales.”
NLS.LC.MONETARY	Each record in this file holds the monetary formatting convention used in a locale. For a description of the record format for this file, see Format of Convention Records in Chapter 4, “Locales.”
NLS.LC.NUMERIC	Each record in this file holds the numeric formatting convention used in a locale. For a description of the record format for this file, see Format of Convention Records in Chapter 4, “Locales.”
NLS.LC.TIME	Each record in this file holds the time and date formatting convention for a locale. For a description of the record format for this file, see Format of Convention Records in Chapter 4, “Locales.”

NLS Database Files (Continued)

File	Description
NLS.MAP.DESCS	Contains descriptions of every map known to UniVerse. The record ID of each map is the map name used in UniVerse commands or UniVerse BASIC programs. The record IDs must comprise ASCII-7 characters only. For a description of the record format for this file, see Creating a Map Description in Chapter 3, “Maps.”
NLS.MAP.TABLES	A type 19 file that contains the map tables for mapping an external character set to the UniVerse internal character set. For more information about the structure of this file, see Creating a Map Table in Chapter 3, “Maps.”
NLS.WT.LOOKUP	Contains weightings given to characters during a sort, based on the Unicode standard. This file should not be modified.
NLS.WT.TABLES	Contains specific weight information about characters used in a locale. For more information about the structure of this file, see Editing Weight Tables in Chapter 4, “Locales.”

NLS Database Files (Continued)

National Convention Hooks

The national conventions support described in the *UniVerse NLS Guide* does not cover all needs. It is designed to be as table-driven as possible, with all tables visible to and changeable by a knowledgeable user. For maximum flexibility, we also support user-written code *hooks*. These are routines you write to implement specific NLS functions and then hook them into UniVerse on request.

Hooks are points in UniVerse code where an NLS convention is in force; at such points, user-written code can be plugged in to intercept an action that NLS would otherwise perform. Hook routines must be written in C. Each routine has a fixed name and interface, as described later.

All string data is passed in and out of hooks in external format (for example, as multibyte 8-bit strings). That is, a map name (other than UNICODE) associated with a hook is used to map string data from UniVerse internal format to external format before calling the hook. All hooks for a particular locale specify the same map name. To accommodate CHAR(0) bytes, STRING data types are used (a variable-length character string) rather than null-terminated C strings.

This hook mechanism is available only if both NLS mode and locale support are enabled. The hooks also introduce some areas of potential internationalization that are not otherwise supported by NLS, notably:

- Specialized FMT format codes
- Soundex ‘sounds-like’ replacement

General Hook Mechanism

You write C code conforming to the naming and calling conventions described later, and link them into UniVerse using the tools described. You then set up a locale record in which the `HOOK_LIBRARY_ID` and `HOOK_MAPNAME` fields are filled in. These identify which hook library to invoke if you set that locale, and what map to use to convert strings to external form. The hook library must contain an `ih_init` function and whatever other `ih_...` functions it wants to implement.

When the `SET.LOCALE` command or the `SETLOCALE` function invokes the locale, its `HOOK_LIBRARY_ID` is used to call the appropriate `ih_init` function. From now on, assume that `HID` is the specific “hook_locale_id”, which can be any alphanumeric string (for example, `GB`, `HEBREW`, and so forth). So UniVerse tries to call `ih_init_HID`. If there is no such function linked, the locale cannot be set. Otherwise, `ih_init_HID` returns a list of which other hook routines are included in the library. This information is then used elsewhere in UniVerse where conventions would apply. If the hook in question is set, UniVerse tries to call the appropriate `ih_xxx_HID` function.

For ease of implementation, the file `sample/NLSHKtmpl.c` in the UV account directory provides a complete set of stub routines. Copy this file to create a hook library of your own, replacing all `HID` suffixes with your chosen `HID`. Then add code to the hooks you want to implement, and change `ih_init_HID` so it says which functions contain real code. This avoids unnecessary complication in the linking mechanism (since all functions of a library exist, even if empty), but also stops the performance overhead of calling empty functions. Basically, the UniVerse code makes a call to `ih_xxx_HID` only if the current locale says it’s worth it. The use of `HID` lets you develop multiple hook libraries independently and then link them together easily—all it requires is that they choose different `HIDs`.

The following routines can be in a hook library:

```
ih_case_HID()  
ih_compare_HID()  
ih_ctype_HID()  
ih_fmt_HID()  
ih_iconv_HID()  
ih_lendp_HID()  
ih_match_HID()  
ih_oconv_HID()  
ih_soundex_HID()  
ih_trim_HID()
```

Each hook has a similar form. There are usually `in_str`, `out_str`, and `replaced_char` arguments. All the functions return an integer value with the same basic meaning.

```
int ih_xxx_HID(String in_str, int replaced_char, String *out_str, ...  
other args)
```

Parameter	Description
<code>in_str (I)</code>	Incoming data in external format as mapped by the <code>HOOK_MAPNAME</code> map (which defaults to the value in the <code>uvconfig</code> parameter <code>NLSOSMAP</code>).
<code>replaced_char (I)</code>	Set to 1 if <code>in_str</code> contains a character that had to be mapped to the replacement character in the external set.
<code>out_str (O)</code>	Pointer to the <code>STRING</code> structure containing outgoing data in the same external character set. Only valid if the returned value is <code>NLSHK_HKE_OK</code> or <code>NLSHK_HKE_SOME_CONV</code> .

ih_xxx_HID Parameters

The returned value can be one of the following:

Value	Description
NLSHK_HKE_OK	Hook routine did its stuff, no further action required by UniVerse—use what is in out_str.
NLSHK_HKE_NO_CONV	Hook routine did nothing, UniVerse pretends it was never called and continues processing the in_str.
NLSHK_HKE_SOME_CONV	Hook routine did something but wants UniVerse to continue its own processing, using the contents of out_str rather than in_str. The data is first mapped back to internal form.

ih_XXX_HID Return Values

The ih_XXX_HID routines in the *sample/NLSHKtmpl.c* file ignore all input arguments and simply return NLSHK_HKE_NO_CONV.

Support from UniVerse

The `STRING` data type used in the interface definitions requires the UniVerse file `gcidir/include/uv.h` in the UV account directory to be included in the source of the hook library.

Also required is the UniVerse include file `gcidir/include/flavor.h`, which contains definitions for the account flavor types used by `ih_fmt_HID`, `ih_iconv_HID`, and `ih_oconv_HID`.

The NLS hook table must be initialized by the `ih_init_HID` routine. The NLS hook table is a global structure, a reference to which can be found in the public include file `gcidir/include/NLShooks.h`. Include this file in your hook library source files.

Memory Management

Hook routines are responsible only for the memory they allocate to perform their allotted function, for example, memory for return parameters and temporary variables. They do not need to worry about memory occupied by input parameters; UniVerse deals with this.

Memory must be allocated and freed using the standard system memory allocator interfaces: *malloc* (and *realloc*) to allocate memory and *free* to deallocate it.

Using Hooks in UniVerse

To make UniVerse use a hook library, complete the following steps:

1. Create a GCI definition for the initialization routine.
2. Compile the hook library.
3. Build the hook library.
4. Test the hooks.
5. Install the hook library.

Since the GCI identifies the hook library to UniVerse, and since the GCI differs slightly on the UNIX and Windows versions of UniVerse, there are a few differences in the steps required on both platforms. See the *UniVerse GCI Guide* for details on how to use the GCI.

In the examples shown, a set of hook library routines is written in the file *my_hooks.c*.

Create a GCI Definition for the Initialization Routine

On Windows platforms only, you must create a GCI definition file (GCI menu, option 1) to hold the definition of the hook library initialization routine. Remember the name of the file: you will need it when you build the hook library.

Then, on both platforms, choose the GCI menu option to add a GCI subroutine definition (GCI menu, option 1 on UNIX, option 2 on Windows platforms). On Windows platforms, use the GCI definition file you just created.

The purpose is to add a definition for the initialization routine `ih_init_HID`, as follows:

Attribute	Definition
Subroutine name:	<code>ih_init_HID</code>
Language:	C
External name:	<code>ih_init_HID</code>
Module name:	<code>my_hooks</code> <--- i.e. <code>my_hooks.c</code>

ih_init_HID Initialization Routine

Attribute	Definition
Description:	My hooks
Number of args:	0
Return value:	void

ih_init_HID Initialization Routine (Continued)

HID is the hook library identifier, for example, HEBREW. This identifies the initialization routine to UniVerse and lets it be called.

Compile the Hook Library

Check that the hook library source file (for example, *my_hooks.c*) compiles, and put a copy of the source file in the GCI directory *gcidir* of the UV account directory.

Build the Hook Library

Build the hook library (on UNIX, GCI menu option 4, Make a new UniVerse, on Windows platforms, option 5, Make a GCI Library from a GCI Definition File). This will ultimately compile the hook library source file. On Windows platforms, remember to specify the name of the GCI definition file that you created.

On UNIX the hook library object file produced by the compilation will also be linked with UniVerse to produce a new UniVerse executable, *uvsh.new*, in the UV account directory.

On Windows platforms, the sequence of events is slightly different. The result of menu option 5 is a dynamic link library (DLL) in the GCI directory that has the same name as the GCI definition file that you created.

Test the Hooks

Before you can use the hooks, you must create a locale with the HOOK_LIBRARY_ID and HOOK_MAPNAME fields set appropriately. Do this from the NLS.ADMIN menu in the UV account. The HOOK_LIBRARY_ID must be the same as the HID suffix given to the hook routines. The hooks are designed to work with a given character set, so set the HOOK_MAPNAME to the corresponding NLS map name for this character set.

To use the hook library routines, the NLSLCMODE parameter in the *uvconfig* file must be set to 1 and UniVerse must be reloaded (on UNIX, *uvregen* followed by *DBsetup*, on Windows platforms, *uvregen* followed by stopping and restarting all of the services).

On UNIX, start UniVerse using the *uvsh.new* executable. The run file is stand-alone and will not disrupt other users.

On Windows platforms, set up the environment variable UVGCIDLLS to include the path of the DLLs generated by the build in the *gcidir* directory (or add the paths to the system variable UvGCILibraries), then start up UniVerse using the *bin\uvsh* executable.

When this is done, the hooks can be activated using the SET.LOCALE command or the SETLOCALE BASIC function. This executes the *ih_init_HID* routine and makes the hooks ready for use.

Install the Hook Library

When testing is complete, you can install the hook library on a UNIX system using the GCI menu, option 5, Install new UniVerse, and on a Windows system, option 6, Install a GCI Library. This makes the hooks available to all users on the system without having to change anything in their environment.

NLS Hook Interface Definitions

Here are a few general rules regarding hook functions:

- A hook function should not free any strings passed to it.
- A hook function is called only if the corresponding UniVerse BASIC statement is executed. For example, the hook for `iconv` is called only if a UniVerse BASIC program calls `ICONV`. If an internal function of UniVerse calls `iconv`, the hook function does not execute, as is the case with SQL DML functions.
- If a hook function returns `NLSHK_HKE_NO_CONV`, it should not return any allocated memory.
- All `NLSHK_xxx` tokens are in the include file `gcidir/include/NLShooks.h` in the UV account directory.

Hook Functions

The initialization function `ih_init_HID` initializes each element of the Hook table to a corresponding hook function or sets it to null as shown below. You should replace `HID` with your hook library ID. In the example only the `CASE` hook is supplied.

```
void ih_init_HID()
{
  NLSHKHookTable [NLSHK_TABLE_CASE] = ih_case_HID;
  NLSHKHookTable [NLSHK_TABLE_COMPARE] = 0;
  NLSHKHookTable [NLSHK_TABLE_CTYPE] = 0;
  NLSHKHookTable [NLSHK_TABLE_FMT] = 0;
  NLSHKHookTable [NLSHK_TABLE_ICONV] = 0;
  NLSHKHookTable [NLSHK_TABLE_LENDP] = 0;
  NLSHKHookTable [NLSHK_TABLE_MATCH] = 0;
  NLSHKHookTable [NLSHK_TABLE_OCONV] = 0;
  NLSHKHookTable [NLSHK_TABLE_SOUNDEX] = 0;
  NLSHKHookTable [NLSHK_TABLE_TRIM] = 0;
}
```

The case hook function is called in response to a UniVerse BASIC call to `DOWNCASE` or `UPCASE`. When `ICONV` or `OCONV` is called with a code of lowercase or uppercase, the `CASE` hook function is not called. The hook function must be defined as follows:

Argument	Definition
int	<code>ih_case_HID(in_str, replaced_char, out_str, conv_type)</code>
STRING	<code>in_str;</code>
int	<code>replaced_char;</code>
STRING	<code>*out_str;</code>
int	<code>conv_type;</code>

CASE Hook Function Definition

Argument	Description
in_str	The input string.
replaced_char	Set to 1 if a character was replaced in in_str.
out_str	Output STRING variable whose text field is malloc'd by the hook function if the hook function returns NLSHK_HKE_OK or NLSHK_HKE_SOME_CONV.
conv_type	Input argument to contain NLSHK_CT_DOWNCASE or NLSHK_CT_UPCASE.

CASE Hook Function Argument Description

The hook function's return value should be:

Return Value	Description
NLSHK_HKE_NO_CONV	No conversion done by hook.
NLSHK_HKE_OK	Complete conversion done by hook.
NLSHK_HKE_SOME_CONV	Some conversion done by hook.

CASE Hook Function Return Values

If the hook function returns an invalid value, UniVerse issues a warning.

The compare hook function is called in response to a call to:

- The BASIC COMPARE function
- Simple comparisons of the type <, =, >, LE, GE, NE
- Vector comparisons like LES, LTS, GTS, GES, EQS, NES

The hook function must be defined as follows:

Argument	Value
int	ih_compare_HID(in_str1, rep_char1, in_str2, rep_char2, type, justprec, pretval)
STRING	in_str1;
int	rep_char1;
STRING	in_str2;
int	rep_char2;
int	type, justprec, *pretval;
{	
	*pretval = 0;
	...

Compare Hook Function Definition

Argument	Description
in_str1	The first input string.
rep_char1	Set to 1 if a character was replaced in in_str1.
in_str2	The second input string.
rep_char2	Set to 1 if a character was replaced in in_str2.
type justprec	Input arguments to contain the following values while pretval is an output argument: n COMPARE type is NLSHK_CO_COMPARE. justprec contains 0 for left justification (default), 1 for right justification.

Compare Hook Function Argument Description

Argument	Description
	<p>n Simple comparisons of the type <, =, >, LE, GE, NE type is one of NLSHK_CO_GREATER, NLSHK_CO_GTEQUAL, NLSHK_CO_EQUAL, NLSHK_CO_NEQUAL, NLSHK_CO_LTEQUAL, or NLSHK_CO_LESSTHAN depending on the type of comparison being performed. justprec is the current precision (if required) or 0.</p> <p>n Vector comparisons like LES, LTS, GTS, GES, EQS, NES. type is one of NLSHK_CO_GREATER, NLSHK_CO_GTEQUAL, NLSHK_CO_EQUAL, NLSHK_CO_NEQUAL, NLSHK_CO_LTEQUAL, or NLSHK_CO_LESSTHAN depending on the type of comparison being performed. justprec is the current precision (if required) or 0.</p>
pretval	<p>Must be set to one of the following if the return value is NLSHK_HKE_OK: <0If in_str1 is less than in_str2 0If in_str1 and in_str2 are equal >0If in_str1 is greater than in_str2</p>

Compare Hook Function Argument Description (Continued)

The hook function's return value should be NLSHK_HKE_NO_CONV or NLSHK_HKE_OK. If the hook function returns an invalid value, UniVerse issues a warning.

The ctype hook function is called in response to a call to the UniVerse BASIC function ALPHA, which checks whether a string is alphabetic. The hook function must be defined as follows:

Argument	Value
int	ih_ctype_HID(in_str, replaced_char, pretval)
STRING	in_str;
int	replaced_char;

ctype Hook Function

Argument	Value
int	*pretval;
{	
	*pretval = 0;
	...

ctype Hook Function (Continued)

Argument	Description
in_str	The input string.
replaced_char	Set to 1 if a character was replaced in in_str.
pretval	Must be set to one of the following if the return value is NLSHK_HKE_OK:
1	If in_str is alphabetic
0	If in_str is not alphabetic

ctype Hook Function Argument Description

The hook function's return value should be NLSHK_HKE_NO_CONV or NLSHK_HKE_OK. If the hook function returns an invalid value, UniVerse issues a warning.

The match hook function is called in response to a call to the UniVerse BASIC function MATCH or MATCHFIELD, which check for the presence of a pattern in a string. The hook function must be defined as follows:

Argument	Value
int	ih_match_HID(in_str1, rep_char1, mask_str, rep_char2, out_str, fieldnum, pmatched)
STRING	in_str1;
int	rep_char1;
STRING	mask_str;

Match Hook Function Definition

Argument	Value
int	rep_char2;
STRING	*out_str;
int	fieldnum;
int	*pmatched;
{	
	*pmatched = 0;
	...
}	

Match Hook Function Definition (Continued)

Argument	Description
in_str1	The input string.
rep_char1	Set to 1 if a character was replaced in in_str1.
mask_str	The mask to use.
rep_char2	Set to 1 if a character was replaced in mask_str.
out_str	Output STRING variable whose text field is malloc'd by the hook function in certain cases.
fieldnum	0 if the hook function is for MATCH, otherwise the field number specified to the MATCHFIELD function.
pmatched	Output parameter that indicates whether a match was found (see below).

Match Hook Function Argument Description

The hook function's return value should be NLSHK_HKE_NO_CONV, NLSHK_HKE_OK or NLSHK_HKE_SOME_CONV. If the hook function returns an invalid value, UniVerse issues a warning.

If the hook function is for MATCH:

- If the return value is NLSHK_HKE_NO_CONV, the pmatched argument is irrelevant. out_str should not be set.
- If the return value is NLSHK_HKE_SOME_CONV, the pmatched argument is irrelevant. The hook function should set out_str to contain the relevant output.
- If the return value is NLSHK_HKE_OK, the hook function should set the pmatched argument (1 if pattern found, 0 otherwise). out_str should not be set.

If the hook function is for MATCHFIELD:

- If the return value is NLSHK_HKE_NO_CONV, the pmatched argument is irrelevant. out_str should not be set.
- If the return value is NLSHK_HKE_SOME_CONV, the pmatched argument is irrelevant. The hook function should set out_str to contain the relevant output.
- If the return value is NLSHK_HKE_OK, the pmatched argument is irrelevant. The hook function should set out_str to contain the relevant output.

The format hook function is called in response to a call to the UniVerse BASIC functions FMT and FMTS. The hook function must be defined as follows:

Argument	Definition
int	ih_fmt_HID(in_str, replaced_char, out_str, fmt_code, options_flag, precision)
STRING	in_str;
int	replaced_char;
STRING	*out_str;
STRING	fmt_code;
int	options_flag;
int	precision;

Format Hook Function Definition

Argument	Description
<code>in_str</code>	The input string.
<code>replaced_char</code>	Set to 1 if a character was replaced in <code>in_str</code> .
<code>out_str</code>	Output STRING variable whose text field is malloc'd by the hook function if the hook function's return value is <code>NLSHK_HKE_OK</code> or <code>NLSHK_HKE_SOME_CONV</code> .
<code>fmt_code</code>	Input argument to contain the format code supplied to <code>FMT</code> or <code>FMTS</code> .
<code>options_flag</code>	Input argument to contain one of the following: <code>IDEAL_FLAVOR</code> , <code>PICK_FLAVOR</code> , <code>INFO_FLAVOR</code> , <code>REAL_FLAVOR</code> , <code>IN2_FLAVOR</code> , <code>PIOPEN_FLAVOR</code> Also, if <code>fmt_code</code> is in display positions, the <code>options_flag</code> is ORed with <code>DP_FLAVOR</code> . See the file <code>gcidir/include/flavor.h</code> for these tokens.
<code>precision</code>	The current UniVerse precision.

Format Hook Function Argument Description

The hook function's return value should be `NLSHK_HKE_NO_CONV`, `NLSHK_HKE_OK`, `NLSHK_HKE_SOME_CONV`, `NLSHK_HKE_CC_INVALID` or `NLSHK_HKE_INPUT_INVALID`. `NLSHK_HKE_CC_INVALID` can be used to indicate an invalid conversion code and `NLSHK_HKE_INPUT_INVALID` to indicate invalid data was input for formatting. If the hook function returns an invalid value, UniVerse issues a warning.

The `iconv` and `oconv` hook functions are called in response to a call to the BASIC functions `ICONV`, `OCONV`, `ICONVS`, or `OCONVS`. The hook function must be defined as follows:

Argument	Definition
<code>int</code>	<code>ih_iconv_HID(in_str, replaced_char, out_str, conv_code, options_flag)</code>
<code>int</code>	<code>ih_oconv_HID(in_str, replaced_char, out_str, conv_code, options_flag)</code>
STRING	<code>in_str;</code>

iconv and oconv Hook Definition

Argument	Definition
int	replaced_char;
STRING	*out_str;
STRING	conv_code;
int	options_flag;

iconv and oconv Hook Definition (Continued)

Argument	Description
in_str	The input string.
replaced_char	Set to 1 if a character was replaced in in_str.
out_str	Output STRING variable whose text field is malloc'd by the hook function if the hook function's return value is NLSHK_HKE_OK or NLSHK_HKE_SOME_CONV.
conv_code	Input argument to contain the conversion code to apply.
options_flag	Input argument to contain one of the following: IDEAL_FLAVOR, PICK_FLAVOR, INFO_FLAVOR, REAL_FLAVOR, IN2_FLAVOR, PIOPEN_FLAVOR See the file <i>gcidir/include/flavor.h</i> for these tokens.

iconv and oconv Hook Argument Description

The hook function's return value should be NLSHK_HKE_NO_CONV, NLSHK_HKE_OK, NLSHK_HKE_SOME_CONV, NLSHK_HKE_CC_INVALID or NLSHK_HKE_INPUT_INVALID. NLSHK_HKE_CC_INVALID can be used to indicate an invalid conversion code and NLSHK_HKE_INPUT_INVALID to indicate invalid data was input for formatting. If the hook function returns an invalid value, UniVerse issues a warning.

The lendp hook function is called in response to a call to the UniVerse BASIC functions LENDP and LENS DP. The hook function must be defined as follows:

Argument	Definition
int	ih_lendp_HID(in_str, replaced_char, pretval)
STRING	in_str;
int	replaced_char;
int	*pretval;
{	
	*pretval = 0;
	...
}	

lendp Hook Function Definition

Argument	Description
in_str	The input string.
replaced_char	Set to 1 if a character was replaced in in_str.
pretval	Must be set to the length in display positions of the input string when the return value is NLSHK_HKE_OK.

lendp Hook Function Argument Description

The hook function's return value should be NLSHK_HKE_NO_CONV or NLSHK_HKE_OK. If the hook function returns an invalid value, UniVerse issues a warning.

The soundex hook function is called in response to a call to the UniVerse BASIC function SOUNDEX. The hook function must be defined as follows:

Argument	Definition
int	ih_soundex_HID(in_str, replaced_char, out_str)
STRING	in_str;
int	replaced_char;
STRING	*out_str;

soundex Hook Function Definition

Argument	Description
in_str	The input string.
replaced_char	Set to 1 if a character was replaced in in_str.
out_str	Output STRING variable whose text field is malloc'd by the hook function if the hook function returns NLSHK_HKE_OK or NLSHK_HKE_SOME_CONV.

soundex Hook Function Description

The hook function's return value should be NLSHK_HKE_NO_CONV, NLSHK_HKE_OK, or NLSHK_HKE_SOME_CONV. If the hook function returns an invalid value, UniVerse issues a warning.

The trim hook function is called in response to a call to the UniVerse BASIC functions TRIM, TRIMB, TRIME, TRIMS, TRIMBS, and TRIMFS. For TRIM, the hook function is called only if *expression* is the sole argument specified in the TRIM function call (see *UniVerse BASIC* for more details). The hook function must be defined as follows:

Argument	Definition
int	ih_trim_HID(in_str, replaced_char, out_str, trim_type)
STRING	in_str;
int	replaced_char;

trim Hook Function Definition

Argument	Definition
STRING	*out_str;
int	trim_type;

trim Hook Function Definition (Continued)

The hook function's return value should be NLSHK_HKE_NO_CONV, NLSHK_HKE_OK, or NLSHK_HKE_SOME_CONV. If the hook function returns an invalid value, UniVerse issues a warning.

NLS Quick Reference

This appendix contains reference tables for NLS, including the following:

- UniVerse commands that are available only in NLS mode
- UniVerse commands that support NLS features
- Useful NLS commands
- UniVerse BASIC functionality that is available only in NLS mode
- Map tables that are supplied with NLS
- UniVerse locales
- Unicode blocks

UniVerse Commands

The following table lists UniVerse commands that are available only in NLS mode.

Command	Description
GET.FILE.MAP	Displays the map name associated with the specified file.
GET.LOCALE	Retrieves the current locale settings.
LIST.LOCALES	Lists the current locales.
LIST.MAPS	Lists maps that are built and installed in shared memory.
NLS.UPDATE.ACCOUNT	Updates an account to NLS mode.
RESTORE.LOCALE	Restores a locale.
SAVE.LOCALE	Saves a locale.
SET.FILE.MAP	Associates a map name with a file.
SET.GCI.MAP	Sets a map for passing character string parameters to and from GCI subroutines.
SET.LOCALE	Sets or restores a locale.
SET.SEQ.MAP	Associates a map with sequential I/O.
UNICODE.FILE	Converts a mapped file to the UniVerse internal character set, or vice versa, without copying the file.

Commands Available Only in NLS Mode

The next table lists UniVerse commands that behave differently in NLS mode.

Command	Description
ANALYZE.FILE	Reports the map name on a file.
ASSIGN	Defines a map name for an assigned tape device.
BASIC	Optionally sets a map during compilation using the \$MAP compiler directive.

Commands That Change in NLS Mode

Command	Description
COPY, CP, and CT	Have a UNICODE keyword that prints each character as a Unicode 4-digit hexadecimal value. The HEX keyword displays internal hexadecimal character values.
CREATE.FILE	Adds a map name to a file as specified in the NLSNEWFILEMAP and NLSNEWDIRMAP parameters in the <i>uvconfig</i> file.
ED	The ED command has an extended up-arrow mode for undisplayable multinational characters.
FILE.STAT	Reports the map name on a file.
GET.TERM.TYPE	Reports the name of the terminal or auxiliary printer map.
SETPTR (UNIX), SETPTR (Windows Platforms)	Optionally associates a map with a print channel in order to determine display widths for formatting spooled output.
SET.TERM.TYPE	Sets a map for a terminal or auxiliary printer.
T.ATT	Defines a map name for an assigned tape device.
TERM	Reports the name of the terminal or auxiliary printer map.

Commands That Change in NLS Mode (Continued)

The next table contains other useful NLS commands.

Command	Description
EDIT.CONFIG	Edits the <i>uvconfig</i> file. This command is also available by choosing Installation -> Edit uvconfig from the NLS Administration menu.
NLS.ADMIN	Enters the NLS Administration menu system.

Useful NLS Commands

BASIC Statements and Functions

The following table lists BASIC statements and functions that provide new functionality when NLS is enabled.

Statement/Function	Description
AUXMAP statement	Switches to a terminal's auxiliary map.
FILEINFO function	Returns a file's map name.
FMTDP function	Formats a string in display positions rather than character positions. If NLS mode is off, FMTDP acts like FMT.
FMTSDP function	Formats a dynamic array in display positions rather than character positions. If NLS mode is off, FMTSDP acts like FMTS.
FOLDDP function	Determines where to fold a string using display positions. If NLS mode is off, FOLDDP acts like FOLD.
FOOTING statement	Calculates gaps in footings using display positions.
GETLOCALE function	Retrieves the names of specified categories of the current locale.
HEADING statement	Calculates gaps in headings using display positions.
ICONV function	Uses the NLS, MU0C, other new conversion codes.
INPUTDP statement	Defines input formats using display positions.
LENDP function	Returns the length of a string in display positions. If NLS mode is off, LENDP acts like LEN.
LENSDP function	Returns the length of a dynamic array in display positions. If NLS mode is off, LENS DP acts like LEN.
LOCALEINFO function	Retrieves the settings of the current locale.
ONCONV function	Uses the NLS, MU0C, and other new conversion codes.
SETLOCALE function	Changes the setting of one or all categories for the current locale.
STATUS function	Returns additional values for READ and WRITE statements that encounter unmappable characters.

BASIC Functionality Available Only in NLS Mode

Statement/Function	Description
SYSTEM function	Returns a value to indicate the current NLS mode and other NLS parameters.
UNICHAR function	Generates a single character in external format.
UNICHARS function	Generates a dynamic array in external format.
UNISEQ function	Returns the Unicode value of a single character in internal format.
UNISEQS function	Returns a dynamic array of Unicode values in internal format.
UPRINT statement	Sends data to a printer without using the printer's map.
!GETPU	Determines the map name associated with a print channel.

BASIC Functionality Available Only in NLS Mode (Continued)

Map Tables

The following list shows all the map tables for major character sets used worldwide that are supplied with UniVerse. The left column contains the name of the map, the middle column contains the name of the map table used by the map (in NLS.MAP.TABLES), and the right column contains a description of the map.

MAP.DESCS..... description.....	Table ID.....	Map
ASCII	ASCII	#Standard ASCII 7-bit set
ASCII+C1	ASCII	ASCII 7-bit + C1 control chars
ASCII+MARKS marks	UV-MARKS	#Std ASCII 7-bit set for type 1&19 files w/
BIG5	BIG5	#TAIWAN: "Big 5" standard
C0-CONTROLS 1F+7F	C0-CONTROLS	Standard ISO2022 C0 control set, chars 00-
C1-CONTROLS	C1-CONTROLS	Standard 8-bit ISO control set, 80-9F
EBCDIC	EBCDIC	#IBM EBCDIC as implemented by standard
uniVerse		- full set
EBCDIC-037	EBCDIC-037	#IBM EBCDIC variant 037
EBCDIC-1026	EBCDIC-1026	#IBM EBCDIC variant 1026 (Turkish)
EBCDIC-500V1	EBCDIC-500V1	#IBM EBCDIC variant 500V1
EBCDIC-875	EBCDIC-875	#IBM EBCDIC variant 875 (Greek)
EBCDIC-CTRLS	EBCDIC-CTRLS	IBM EBCDIC as implemented by standard uniVerse - control chars only
GB2312	GB2312-80	#CHINESE: EUC as described by GB 2312
ISO8859-1	ISO8859-1	#Standard ISO8859 part 1: Latin-1
ISO8859-1+MARKS	UV-MARKS	#Standard ISO8859 part 1: Latin-1 for type 1& 19 files with marks
ISO8859-10	ISO8859-10	#Standard ISO8859 part 10: Latin-6
ISO8859-2	ISO8859-2	#Standard ISO8859 part 2: Latin-2
ISO8859-3	ISO8859-3	#Standard ISO8859 part 3: Latin-3
ISO8859-4	ISO8859-4	#Standard ISO8859 part 4: Latin-4
ISO8859-5	ISO8859-5	#Standard ISO8859 part 5: Latin-Cyrillic
ISO8859-6	ISO8859-6	#Standard ISO8859 part 6: Latin-Arabic
ISO8859-7	ISO8859-7	#Standard ISO8859 part 7: Latin-Greek
ISO8859-8	ISO8859-8	#Standard ISO8859 part 8: Latin-Hebrew
ISO8859-9	ISO8859-9	#Standard ISO8859 part 5: Latin-5
JIS-EUC	JISX0208	#JAPANESE: EUC excluding JIS X 0212 Kanji
JIS-EUC+	JISX0212	#JAPANESE: EUC including JIS X 0212 Kanji
JIS-EUC-HWK	JISX0201-K	JAPANESE: 1/2 width katakana for JIS-EUC
JIS-EUC2	JISX0208	#JAPANESE: EUC fixed width excluding JIS X 02 12 kanji
JIS-EUC2+	JISX0212	#JAPANESE: EUC fixed width including JIS X 02 12 kanji
JIS-EUC2-C0	C0-CONTROLS	JAPANESE: EUC2 fixed width C0 control chars
JIS-EUC2-C1	C1-CONTROLS	JAPANESE: EUC fixed width C1 control chars
JIS-EUC2-HWK	JISX0201-K	JAPANESE: EUC fixed width representation of 1 /2 width katakana
JIS-EUC2-MARKS	JIS-EUC2-MARKS	JAPANESE: EUC2 fixed width mark characters (e xternal form)
JIS-EUC2-ROMAN	JISX0201-A	JAPANESE: EUC fixed width representation of J IS-ROMAN
JIS-ROMAN	JISX0201-A	#JAPANESE: Variant of 7-bit ASCII

JISX0201	JISX0201-K	#JAPANESE: Single-byte set, 1/2 width katakana + ASCII
KOI8-R	KOI8-R	#KOI8-R Russian/Cyrillic set
KSC5601	KSC5601	#KOREAN: Wansung code as described by KS C
5601		-1987
MAC-GREEK	MAC-GREEK	#Apple Macintosh Greek Repertoire (like
ISO8859-7)		
MAC-GREEK2	MAC-GREEK2	#Apple Macintosh Greek Repertoire based on
APPLE II		
MAC-ROMAN	MAC-ROMAN	#Apple Macintosh Roman character set, based on
		ASCII
MNEMONICS		#ASCII mnemonics for many Unicodes, based on
		UTF8
MNEMONICS-1	ISO8859-1	#As for MNEMONICS, but ISO8859-1 capable
MS1250	MS1250	#MS Windows code page 1250 (Latin 2)
MS1251	MS1251	#MS Windows code page 1251 (Cyrillic)
MS1252	MS1252	#MS Windows code page 1252 (Latin 1)
MS1253	MS1253	#MS Windows code page 1253 (Greek)
MS1254	MS1254	#MS Windows code page 1254 (Turkish)
MS1255	MS1255	#MS Windows code page 1255 (Hebrew)
MS1256	MS1256	#MS Windows code page 1256 (Arabic)
PC1040	PC1040	#PC DOS code page 1040 (Korean)
PC1041	PC1041	#PC DOS code page 1041 (Japanese)
PC437	PC437	#PC DOS code page 437 (US)
PC850	PC850	#PC DOS code page 850 (Latin 1)
PC852	PC852	#PC DOS code page 852 (Latin 2)
PC855	PC855	#PC DOS code page 855 (Cyrillic)
PC857	PC857	#PC DOS code page 857 (Turkish)
PC860	PC860	#PC DOS code page 860 (Portuguese)
PC861	PC861	#PC DOS code page 861 (Icelandic)
PC863	PC863	#PC DOS code page 863 (Canada-Fr)
PC864	PC864	#PC DOS code page 864 (Arabic)
PC865	PC865	#PC DOS code page 865 (Nordic)
PC866	PC866	#PC DOS code page 866 (Cyrillic)
PC869	PC869	#PC DOS code page 869 (Greek)
PIECS	PIECS	#PI and PI/open Extended Character Set
PRIME-SHIFT-JIS	PJISX0208	#JAPANESE: Shift-JIS main map (Prime variant)
SHIFT-JIS	SJISX0208	#JAPANESE: Shift-JIS main map
TAU-SHIFT-JIS	TJISX0208	#JAPANESE: Shift-JIS main map (Tau variant)
TIS620	TIS620-A	#THAI: standard TIS 620 ("Thai ASCII")
TIS620-B	TIS620-B	Non-spacing characters part of TIS620 (Thai)

UniVerse Locales

The following list shows the locales supplied with UniVerse, the territory that uses each locale, and the relevant language:

NLS.LC.ALL.....	Description.....
AR-SPANISH	Territory=Argentina, Language=Spanish
AT-GERMAN	Territory=Austria, Language=German
AU-ENGLISH	Territory=Australia, Language=English
BE-DUTCH	Territory=Belgium, Language=Dutch
BE-FRENCH	Territory=Belgium, Language=French
BE-GERMAN	Territory=Belgium, Language=German
BG-BULGARIAN	Territory=Bulgaria, Language=Bulgarian
BO-SPANISH	Territory=Bolivia, Language=Spanish
BR-PORTUGUESE	Territory=Brazil, Language=Portuguese
CA-ENGLISH	Territory=Canada, Language=English
CA-FRENCH	Territory=Canada, Language=French
CH-FRENCH	Territory=Switzerland, Language=French
CH-GERMAN	Territory=Switzerland, Language=German
CH-ITALIAN	Territory=Switzerland, Language=Italian
CL-SPANISH	Territory=Chile, Language=Spanish
CN-CHINESE	Territory=China (PRC), Language=Chinese
CO-SPANISH	Territory=Colombia, Language=Spanish
CR-SPANISH	Territory=Costa Rica, Language=Spanish
CZ-CZECH	Territory=Czech Republic, Language=Czech
DE-GERMAN	Territory=Germany, Language=German
DK-DANISH	Territory=Denmark, Language=Danish
DO-SPANISH	Territory=Dominican Republic, Language=Spanish
EC-SPANISH	Territory=Ecuador, Language=Spanish
EE-ESTONIAN	Territory=Estonia, Language=Estonian
ES-SPANISH	Territory=Spain, Language=Spanish
EV-SPANISH	Territory=El Salvador, Language=Spanish
FI-FINNISH	Territory=Finland, Language=Finnish
FO-FAEROESE	Territory=Faeroe Islands, Language=Faeroese
FR-FRENCH	Territory=France, Language=French
GB-ENGLISH	Territory=UK, Language=English
GL-GREENLANDIC	Territory=Greenland, Language=Greenlandic
GR-GREEK	Territory=Greece, Language=Greek
GT-SPANISH	Territory=Guatemala, Language=Spanish
HN-SPANISH	Territory=Honduras, Language=Spanish
HR-CROATIAN	Territory=Croatia, Language=Croatian
HU-HUNGARIAN	Territory=Hungary, Language=Hungarian
IE-ENGLISH	Territory=Ireland, Language=English
IL-ENGLISH	Territory=Israel, Language=English
IL-HEBREW	Territory=Israel, Language=Hebrew

IS-ICELANDIC	Territory=Iceland, Language=Icelandic
IT-ITALIAN	Territory=Italy, Language=Italian
JP-JAPANESE	Territory=Japan, Language=Japanese
KP-KOREAN (NORTH),	Territory=Democratic People's Republic of Korea Language=Korean
KR-KOREAN	Territory=Republic of Korea (SOUTH), Language=Korean
LT-LITHUANIAN	Territory=Lithuania, Language=Lithuanian
LV-LATVIAN	Territory=Latvia, Language=Latvian
MX-SPANISH	Territory=Mexico, Language=Spanish
NL-DUTCH	Territory=Netherlands, Language=Dutch
NO-NORWEGIAN	Territory=Norway, Language=Norwegian
NZ-ENGLISH	Territory=New Zealand, Language=English
PA-SPANISH	Territory=Panama, Language=Spanish
PE-SPANISH	Territory=Peru, Language=Spanish
PL-POLISH	Territory=Poland, Language=Polish
PT-PORTUGUESE	Territory=Portugal, Language=Portuguese
RO-ROMANIAN	Territory=Romania, Language=Romanian
RU-RUSSIAN	Territory=Russia, Language=Russian
SE-SWEDISH	Territory=Sweden, Language=Swedish
SI-SLOVENIAN	Territory=Slovenia, Language=Slovenian
TR-TURKISH	Territory=Turkey, Language=Turkish
TW-CHINESE	Territory=Taiwan, Language=Chinese
US-ENGLISH	Territory=USA, Language=English
UY-SPANISH	Territory=Uruguay, Language=Spanish
VE-SPANISH	Territory=Venezuela, Language=Spanish
ZA-ENGLISH	Territory=South Africa, Language=English

Unicode Blocks

Unicode is divided into blocks of related characters. These correspond approximately to the scripts used for different families of languages. Characters allocated within blocks have a code value and a description. The description must use uppercase A through Z, hyphen, and digits 0 through 9 only. In UniVerse NLS, the blocks are allocated numbers starting from 1. The main blocks are shown in the following table.

No.	Block Description	Start	End	Usage
1	CONTROL SET 0	0000	001F	ASCII control characters
2	BASIC LATIN	0020	007F	ASCII printing characters
3	CONTROL SET 1	0080	009F	Second control character set from ISO8859- <i>n</i>
4	LATIN-1 SUPPLEMENT	00A0	00FF	Rest of ISO8859-1 (Latin-1) printing characters
5	LATIN EXTENDED-A	0100	017F	Mainly East European, other ISO8859/ <i>n</i>
10	BASIC GREEK	0370	03CF	Greek alphabet, based on ISO8859/7
12	CYRILLIC	0400	04FF	Russian alphabet and related languages
16	BASIC HEBREW	05D0	05EA	Hebrew alphabet, based on ISO8859-8
18	BASIC ARABIC	0600	0652	Based on ISO8859/6
35	THAI	0E00	0E7F	Thai language, based on TIS620
69	CJK SYMBOLS AND PUNCTUATION	3000	303F	For Chinese, Japanese, and Korean
70	HIRAGANA	3040	309F	Japanese syllabary
71	KATAKANA	30A0	30FF	Japanese syllabary

Unicode Blocks

No.	Block Description	Start	End	Usage
97	CJK UNIFIED IDEOGRAPHS	4E00	9FFF	Unification area for Chinese- derived characters
102	HANGUL SYLLABLES	AC00	D7A3	Korean-only characters
107	PRIVATE USE AREA	E000	F8FF	User-defined
116	HALFWIDTH / FULLWIDTH FORMS	FF00	FFEF	Mainly for CJK use

Unicode Blocks (Continued)

Glossary

base map	A character set map upon which another map is based. For example, most character sets use an ASCII map as their base map with additional sets of characters building on the ASCII map.
category	One of the five national conventions: Time, Numeric, Monetary, Collate, or Ctype.
character set	A fixed association between the characters used by a language, or group of languages and the values, or code points , that represent them. For example, the KSC5601 character set fixes code points for the Hangul characters used in the Korean language.
code point	A number that is used in a program to represent a character. Note that in different character sets the same code point may be used to represent different characters.
deadkey characters	Characters that do not have a dedicated key on the keyboard, but are generated using a sequence of key strokes.
deadkey table	See <i>input map table</i> .
doublebyte character set	A character set where the code points are either one or two bytes long. The two-byte code points usually represent characters belonging to Asian languages, such as Chinese or Kanji. See also <i>singlebyte character set</i> .
EBCDIK character set	A variant of the EBCDIC character set. EBCDIK replaces lowercase Latin characters with Japanese Katakana characters.
external character set	The character set used to input data on a keyboard, display data on a screen, print reports, and so on. Appendix C, “NLS Quick Reference” lists the external character sets supported by UniVerse. See also <i>internal character set</i> and <i>Unicode</i> .

JEF character set	A Fujitsu proprietary encoding of several thousand characters. It includes the single-byte EBCDIK and doublebyte JIS character sets. The JEF character set differs from all other character sets that UniVerse NLS supports, in that it uses a pair of shift characters to toggle between singlebyte and doublebyte encoding.
input map table	Mapping tables used to define byte sequences that are valid only on input. They are used to define deadkey characters.
internal character set	The character set that UniVerse uses to store and manipulate data. See also <i>external character set</i> and <i>Unicode</i> .
locale	The language, character set, and data formatting conventions used by a group of people. In UniVerse, a locale comprises a set of conventions in specific categories (Time, Numeric, Monetary, Ctype, and Collate). See also <i>territory</i> .
main map table	The main table that defines how a character set is mapped between the internal and external character sets.
national conventions	A standard set of rules that defines how certain data types such as numbers and dates are used in a territory.
National Language Support (NLS)	See <i>NLS</i> .
NLS	A program's ability to use any languages, data formatting rules, or character sets, that are required by its users all over the world. Also referred to as internationalization.
singlebyte character set	A character set whose code points have values 0 through 255, and can therefore be represented by a single byte. Singlebyte character sets are suitable for some European, American, and Middle Eastern languages. See also <i>doublebyte character set</i> .
territory	The area or region where a locale is used. This may correspond to a geographical location, such as a country, or to something less easy to define in geographical terms, such as a multinational organization.
Unicode	A 16-bit character set that aims to provide unique code points for all characters in every standard character set (with room for some nonstandard characters too). Unicode forms part of ISO 10646 and is a trademark of Unicode, Inc.
Unicode blocks	Groups of logically related characters in the Unicode character set that correspond to the scripts used for different families of languages.

Unicode replacement character	The character value xFFFD, which is used to replace an unmappable character read from the <i>external character set</i> .
unknown character	The character that is used as a substitute for an <i>unmappable character</i> . Each map contains a definition of an unknown character.
unmappable character	A character that cannot be mapped to the external character set using the current map table. UniVerse substitutes the current map's <i>unknown character</i> , usually a question mark (?), for any unmappable character.
UTF8	UTF8 is a standard for the use Unicode character data in 8-bit UNIX environments. In UniVerse UTF8 is enhanced to map the UniVerse system delimiters to the Private Use area of Unicode. Other UTF8-compatible software can understand the UniVerse UTF8 representation.

Index

Numerics

7-bit ASCII 1-5, 5-18
8-bit EBCDIC 5-17

A

accent weight 4-27
accounts
 updating 2-20
adding characters to maps 3-16
alphabetic characters 4-20, 7-3
ANALYZE.FILE command 3-17, C-2
APIs 2-21, 6-3
ASCII function 5-17
ASSIGN command 2-19, C-2
assigning maps 3-17, 5-11
auxiliary devices, setting maps for 5-12
auxiliary printers, setting maps for 2-17, 5-12
AUXMAP statement 5-12, C-4

B

base maps 3-3
 definition GL-1
BASIC
 and locales 5-26
 and multinational characters 5-18
 determining display length 5-7
 determining string length 5-5
 functions and statements C-4
 and maps 5-10
BASIC command C-2
BCI 2-21, 6-3

BCI client programs 6-4, 6-12
block characters
 listing 7-3
block size 5-8
blocks, *see* Unicode: blocks
building
 locales 7-6
 maps 7-5
BYTE function 5-21

C

case hook function B-11
case inversion 5-25
case weight 4-27
Categories menu 7-7
categories, *see* locale categories
changing locale setting 4-34, 5-26
CHAR function 5-18, 5-21
character sets 1-4, 3-3
 code points 1-4
 definition GL-1
 mapping between internal and external 1-4
 maps 1-6
 maps for multibyte 3-14
 Unicode 1-6
characters
 see also Unicode characters
 alphabetic 4-20, 7-3
 defining in Unicode 3-16
 listing Unicode block 7-3
 multibyte 6-15
 nonprinting 7-4
 radix 1-8
 7-bit ASCII 1-5

- storing 1-4
- Characters menu 7-3
- client programs 2-21, 6-3 to 6-16
 - code page 2-22
- code page 2-22
- code point 1-4, 3-3, 5-18
 - definition GL-1
- Collate category 2-12, 4-3, 4-23
 - definition 1-9
- Collate records 4-23
- collating
 - accented sorts 4-23
 - considering case 4-23
 - contractions and expansions 4-29
 - issues 4-27
 - in UniVerse 4-27
- compare hook function B-12
- compiling
 - locales 7-8
 - maps 7-8
- CONFIG command 2-9
- configurable parameters 2-26, 6-4
 - editing 7-8
 - NLSDEFDEVMAP 2-6, 2-19
 - NLSDEFDIRMAP 2-6, 2-16
 - NLSDEFFILEMAP 2-6, 2-16
 - NLSDEFGCIMAP 2-7
 - NLSDEFPTRMAP 2-7, 2-11
 - NLSDEFSEQMAP 2-7
 - NLSDEFSRVLC 2-7, 2-22, 6-4
 - NLSDEFSRVMAP 2-7, 2-22, 6-4
 - NLSDEFTERMMAP 2-7, 2-11, 2-17
 - NLSLCDEF 2-7
 - NLSLCMODE 2-7, 6-4
 - NLSMODE 2-7, 6-4
 - NLSNEWDIRMAP 2-8, 2-16, 3-17
 - NLSNEWFILEMAP 2-8, 2-16, 3-17
 - NLSOSMAP 2-8
 - NLSREADELSE 2-8, 5-15
 - NLSWRITEELSE 2-8
 - setting 2-6, 2-8
 - table of 2-6
- configuration checklist 2-25
- configuring
 - locales 7-8
 - maps 2-10, 7-8
 - NLS 2-5
 - NLS by language 7-8

- console (Windows NT)
 - mapping 2-3
- convention
 - definition 4-3
- convention records 4-8 to 4-26
- conventions 4-4, 4-5
 - creating 4-6
 - national 1-6, 1-7 to 1-9
 - viewing 4-6
- conversion codes 5-21
- conversions, ASCII and EBCDIC 5-17
- converting
 - lowercase to uppercase 7-4
 - uppercase to lowercase 7-4
- converting strings 5-21
- COPY command 5-24, C-3
- CP command 5-24, C-3
- CREATE.FILE command 3-17, C-3
- creating
 - conventions 4-6
 - locale records 7-6
 - locales 4-7
 - map descriptions 3-8
 - map tables 3-10, 7-5
 - maps 3-8
 - new maps 3-4
- cross-referencing
 - locales 7-6
 - map tables 7-5
- CT command 5-24, C-3
- Ctype category 4-3, 4-19, 7-3, 7-4
 - definition 1-9
- ctype hook function B-14
- Ctype records 4-19
- currency symbols
 - international 4-16
 - local 4-16

D

- data types 6-15
- DBsetup* command 2-4
- deadkey characters
 - and case inversion 5-25
 - definition 3-3, GL-1
- deadkey tables 3-3
 - definition GL-1

- decimal places, specifying in monetary formats 4-16
- decimal separators
 - specifying in monetary formats 4-16
 - specifying in numeric formats 4-14
- defining
 - characters as lowercase 4-20
 - characters as uppercase 4-20
 - characters in Unicode 3-16
- DELETE statement 5-5
- deleting
 - locale records 7-6
 - locales 7-8
 - map tables 7-5
 - maps 7-8
- devices, setting maps for 2-14
- digits 7-3
 - specifying alternatives to ASCII 4-14
- disabling locales 4-34, 5-26
- display length
 - when folding strings 5-8
- displaying records in hexadecimal values 5-24
- double-byte character set 3-6
 - definition GL-1

E

- EBCDIC function 5-17
- EBCDIK character set
 - definition GL-1
- ED command 5-18, C-3
- editing
 - configurable parameters 7-8
 - locale records 7-6
 - map tables 7-5
 - maps 3-8
 - multinational characters 5-18
 - uvconfig* file 2-8
 - weight tables 4-30
- EDIT.CONFIG command 2-8, C-3
- enabling
 - locale mode 6-4
 - NLS 1-5, 6-4
- enabling NLS 1-5
- era names 4-10
- EXCHANGE function 5-25

external character sets 1-4, 1-5, 3-3, 3-10, 3-17
 definition GL-1
 EXTRACT function 5-5

F

FIELD function 5-5
 field mark 3-11, 3-15
 FILEINFO function 3-17, C-4
 FILEINFO.H include file 5-11
 files
 FILEINFO.H 5-11
 GETPU.H 5-13
 NLS.CLIENTS.LCS 2-23
 NLS.CLIENT.LCS 7-6, A-2
 NLS.CLIENT.MAPS 2-21, 2-22, 7-5, A-2
 NLS.CS.ALPHAS 4-20, 7-3, A-2
 NLS.CS.BLOCKS A-2
 NLS.CS.CASES 7-4, A-2
 NLS.CS.DESCS 3-13, A-2
 NLS.CS.TYPES 7-3, A-3
 NLS.LANG.INFO 7-7, A-3
 NLS.LC.ALL 4-4, 7-6, A-3
 NLS.LC.COLLATE 2-12, 4-4, 4-23, A-3
 NLS.LC.CTYPE 2-12, 4-4, 4-19, A-3
 NLS.LC.MONETARY 2-12, 4-4, 4-15, A-3
 NLS.LC.NUMERIC 2-12, 4-4, 4-14, A-3
 NLS.LC.TIME 4-4, 4-8, A-3
 NLS.MAP.DESCS 2-12, 3-3, 7-5, A-4
 NLS.MAP.LISTING 3-13
 NLS.MAP.TABLES 2-12, 3-3, 3-10, 7-5, A-4
 NLS.WT.LOOKUP 4-30, 7-7, A-4
 NLS.WT.TABLES A-4
 sp.config 2-14
 type 19 4-30
 UCI configuration 6-9, 6-10
 unmapping 3-18
 uvconfig 2-6, 2-8, 2-11, 2-14, 2-25, 6-9, 6-12, 7-8
 UVNLS.H 5-4

uvodbc.config 6-12, 6-13
 &DEVICE& 2-14
 &HOLD& 2-15
 FILE.STAT command 3-17, C-3
 FINFO\$NLSMAP value of the
 FILEINFO function 5-11
 FMTDP function 5-7, C-4
 FMSTDP function 5-7, C-4
 FOLDDP function C-4
 FOOTING statement C-4
 format hook function B-17
 formatting strings in display
 positions 5-7
 functions
 hook B-11
 InterCall 6-8

G

GCI 2-21, 6-3
 data types 6-15
 specifying maps for 6-15
 subroutines 6-15
 GDI mode 2-14
 GETLOCALE function 5-26, C-4
 GETPU.H include file 5-13
 GET.FILE.MAP command 3-17, C-2
 GET.LOCALE command 4-33, C-2
 GET.TERM.TYPE command 2-18, C-3
 Gregorian calendar 4-10

H

HEADING statement C-4
 HEX option 5-24
 hexadecimal values, displaying records
 in 5-24
 hooks
 functions B-11
 memory management B-6
 national convention B-1
 using in UniVerse B-7

I

ICONV function 5-21, C-4

iconv hook function B-18
 ideographic area (Unicode) 7-3
 include files
 FILEINFO.H 5-11
 GETPU.H 5-13
 UVNLS.H 5-4
 INDEX function 5-5
 input map table, definition GL-2
 input maps 3-3
 INPUT @ statement 5-8
 INPUTDP statement C-4
 inputting
 display positions 5-8
 system delimiters 5-20
 Unicode values 5-20
 INSERT function 5-5
 Installation menu 7-8
 installing
 maps 3-13, 7-8
 NLS 2-3
 InterCall 2-21, 6-3
 functions 6-8
 internal character sets 1-4, 3-3
 definition GL-2
 ipc command 2-4
 ISO 10646 standard 1-4
 ISO 4217 standard 4-16
 item mark 3-11, 3-15

J

Japanese Imperial Era 4-10
 JEF character set
 definition GL-2

L

LEN function 5-5
 LENDP function 5-7, C-4
 lendp hook function B-20
 LENS DP function 5-7, C-4
 listing
 built locales 7-8
 built maps 7-8
 currently installed locales 7-8
 currently installed maps 7-8
 locales 7-6
 map tables 7-5

- maps 7-5
- Unicode block characters 7-3
- Unicode block numbers 7-3
- Unicode characters 7-3
- LIST.LOCALES command C-2
- LIST.MAPS command 3-17, C-2
- locale
 - definition 4-3
- locale categories 1-6, 6-6, 6-7
 - Collate 1-9, 4-3, 4-23
 - Ctype 1-9, 4-3, 4-19
 - definition GL-1
 - Monetary 1-8, 4-3, 4-15
 - Numeric 1-7, 4-3, 4-14
 - Time 1-7, 4-3, 4-8
- locale category
 - definition 4-3
- locale mode
 - enabling 6-4
- locale records
 - creating 7-6
 - deleting 7-6
 - editing 7-6
- LOCALEINFO function C-4
- locales 4-2 to 4-34
 - and BASIC 4-33
 - building 7-6
 - changing current setting of 4-34, 5-26
 - and client programs 2-23
 - compiling 7-8
 - configuring 7-8
 - creating 4-7
 - cross-referencing 7-6
 - definition GL-2
 - deleting 7-8
 - disabling 4-34, 5-26
 - for client programs 6-4, 6-5
 - how they work 4-3
 - listing 7-6
 - listing built 7-8
 - listing installed 7-8
 - moving 2-11
 - naming conventions for 4-7
 - NLS locale configuration program 7-8
 - overview 1-6
 - retrieving settings for 4-33, 5-26
 - saving and restoring 4-33, 5-26

- setting default 2-10
- setting initial 2-13
- supplied with UniVerse C-8
- tables 1-6
 - and UCI 6-9
 - and UniObjects 6-6
 - and UniObjects for Java 6-7
- Locales menu 7-6
- lowercase
 - defining characters as 4-20
 - rules for converting to uppercase 7-4

M

- main map table, definition GL-2
- main maps 3-3
- map descriptions 3-3, 7-5
- map names, where stored 3-17
- map tables 1-5, 1-6, 3-3
 - creating 3-10, 7-5
 - cross-referencing 7-5
 - deleting 7-5
 - editing 7-5
 - listing 7-5
 - table of C-6
- Mappings menu 7-5
- maps 3-2 to 3-18
 - adding characters to 3-16
 - assigning to files 3-17, 5-10
 - and auxiliary printers 2-17
 - base 3-3
 - building 3-13, 7-5
 - character set 1-6
 - for client programs 2-21
 - compiling 7-8
 - configuring 2-10, 7-8
 - creating 3-4, 3-8
 - creating descriptions 3-8
 - deleting 7-8
 - determining current 3-17, 5-10
 - determining for printers 5-13
 - and devices 2-14
 - editing 3-8
 - and existing files 2-16
 - and external character set 3-17
 - for client programs 6-4
 - for multibyte character sets 3-14
 - and GCI 6-15
- getting name 3-17
- and hold files 2-15
- how they work 3-3
- input 3-3
- installing in shared memory 3-13, 7-8
- listing 3-17, 7-5
- listing built 7-8
- listing installed 7-8
- main 3-3
- MNEMONICS 7-3
- modifying 3-13, 3-17
- moving 2-11
- naming conventions 3-6
- and new files 3-17
- NLS map configuration program 7-8
- overview 1-6
 - and sequential files 3-18
- setting default 2-10
- single-byte 3-14
- for source files 5-11
- supplied with UniVerse 1-6, C-6
- and tape devices 2-19
- and terminals 2-17
- and UCI 6-9
- and UniObjects for Java 6-7
- for UNIX pipes 5-13
- unmapping a file 3-18
- and *usp* command 2-15
- for Windows NT console 2-3

- Maps menu 7-8
- mask, inputting display positions
- through 5-8
- match hook function B-15
- MATCHES function 5-5
- memory management and hooks B-6
- menus
- Categories 7-7
- Characters 7-3
- Installation 7-8
- Locales 7-6
- Mappings 7-5
- Maps 7-8
- NLS Administration 2-3, 2-8, 7-2
- Unicode 7-3
- UniVerse System Administration 2-3, 2-4
- MNEMONICS map 7-3
- modifying maps 3-13

Monetary category 4-3, 4-15
 definition 1-8
 Monetary records 4-15
 moving
 locales 2-11
 maps 2-11
 MU0C conversion 5-22
 multibyte character sets 3-14
 multibyte characters 6-15
 and REMOVE pointer 5-9
 in GCI subroutines 6-15
 multibyte Windows NT systems 2-22
 multinational characters
 in BASIC 5-18
 editing 5-18

N

national convention
 definition 4-3
 national conventions 1-6, 1-7 to 1-9, 4-4, 4-5
 definition GL-2
 hooks B-1
 NLS
 and client programs 6-3 to 6-16, 2-21
 configurable parameters 2-6
 configuring by language 7-8
 definition GL-2
 enabling 1-5, 6-4
 installing 2-3
 planning 2-5
 removing 2-4
 shared memory segments 2-4
 updating accounts 2-20
 NLS Administration menu 2-3, 2-8, 7-2
 Build (map) option 2-10, 7-5
 Categories option 7-7
 Installation option 7-8
 Locales option 4-5, 7-6
 Mappings option 2-10, 7-5
 Unicode option 7-3
 NLS conversion code 5-21
 NLS database A-1
nls directory 1-6, A-1
 NLS locale configuration program 7-8

NLS map configuration program 7-8
 NLS mode
 enabling 1-5
 overview 1-3
 testing for 5-4
 NLSDEFDEVMAP parameter 2-6, 2-19
 NLSDEFDIRMAP parameter 2-6, 2-16
 NLSDEFFILEMAP parameter 2-6, 2-16
 NLSDEFGCIMAP parameter 2-7
 NLSDEFPTRMAP parameter 2-7, 2-11
 NLSDEFSEQMAP parameter 2-7
 NLSDEFSRVLC parameter 2-7, 2-22, 6-4
 NLSDEFSRVMAP parameter 2-7, 2-22, 6-4
 NLSDEFTRMMAP parameter 2-7, 2-11, 2-17
 NLSLCDEF parameter 2-7
 NLSLCMODE parameter 2-7, 4-34, 6-4
 NLSLocale object 6-6
 NLSMODE parameter 2-7, 6-4
 NLSNEWDIRMAP parameter 2-8, 2-16, 3-17
 NLSNEWFILEMAP parameter 2-8, 2-16, 3-17
 NLSOSMAP parameter 2-8
 NLSREADELSE parameter 2-8, 5-15
 NLSWRITEELSE parameter 2-8, 5-15
 NLS.ADMIN command C-3
 NLS.CLIENTS.LCS file 2-23
 NLS.CLIENT.LCS file 7-6, A-2
 NLS.CLIENT.MAPS file 2-21, 2-22, 7-5, A-2
 NLS.CS.ALPHAS file 4-20, 7-3, A-2
 NLS.CS.BLOCKS file A-2
 NLS.CS.CASES file 7-4, A-2
 NLS.CS.DESCS file 3-13, A-2
 NLS.CS.TYPES file 7-3, A-3
 NLS.LANG.INFO file 7-7, A-3
 NLS.LC.ALL file 4-4, 7-6, A-3
 NLS.LC.COLLATE file 2-12, 4-4, 4-23, A-3
 NLS.LC.CTYPE file 2-12, 4-4, 4-19, A-3

NLS.LC.MONETARY file 2-12, 4-4, 4-15, A-3
 NLS.LC.NUMERIC file 2-12, 4-4, 4-14, A-3
 NLS.LC.TIME file 4-4, 4-8, A-3
 files
 NLS.LC.TIME 2-12
 NLS.MAP.DESCS file 2-12, 3-3, 7-5, A-4
 NLS.MAP.LISTING file 3-13
 NLS.MAP.TABLES file 2-12, 3-3, 3-10, 7-5, A-4
 NLS.UPDATE.ACCOUNT
 command 2-20, C-2
 NLS.WT.LOOKUP file 4-30, 7-7, A-4
 NLS.WT.TABLES file A-4
 nonprinting characters 7-4
 null value 3-11, 3-15
 system delimiters and 6-5
 using SQLGetInfo 6-11, 6-14
 Numeric category 4-3, 4-14, 7-3
 definition 1-7
 Numeric records 4-14

O

OCONV function 5-21, C-4
 oconv hook function B-18
 ON ERROR clause 5-15
 OPENDEV statement 5-13
 overview
 of locales 1-6
 of maps 1-6
 of NLS mode 1-3
 of Unicode 1-4

P

PICK flavor 5-24
 PRINT statement 5-5
 printing
 in GCI mode 2-14
 in raw mode 2-14
 printing mapped data 5-13
 Private Use area 3-16
 PTERM command 5-25

Q

quick reference, NLS commands C-1 to C-10

R

radix character 1-8, 4-16
 raw mode 2-14
 READ statement 5-5, 5-16
 record IDs, length of 5-6
 REMOVE
 function 5-5
 pointer 5-9
 removing NLS 2-4
 REPLACE function 5-5
 RESTORE.LOCALE command 4-33, C-2
 restoring locales 4-33

S

SAVE.LOCALE command 4-33, C-2
 saving locales 4-33
 secondary indexes, maximum
 characters in 5-6
 SEQ function 5-21
 sequential I/O 3-18, 5-14
 servers 2-21, 6-4
 SETLOCALE function 2-13, 4-33, C-4
 SETPTR command 2-14, C-3
 SETPTR statement 5-7
 SETREM statement 5-9
 setting
 configurable parameters 2-6, 2-8
 default locales 2-10
 default maps 2-10
 initial locale 2-13
 maps for auxiliary printers 2-17, 5-12
 maps for devices 5-12
 maps for tape devices 2-19
 SET.FILE.MAP command 2-16, 3-17, C-2
 SET.GCI.MAP command C-2
 SET.LOCALE command 2-13, 7-8, C-2

SET.SEQ.MAP command 3-18, 5-14, C-2
 SET.TERM.TYPE command 2-17, C-3
 shared memory 3-3
 installing maps in 7-8
 NLS segments 2-4
 shared weight 4-27, 4-28
 single-byte character set 3-6, 3-7
 definition GL-2
 single-byte maps 3-14
 soundex hook function B-21
 source code 5-18
 spool queue directory, how mapped 2-15
sp.config file 2-14
 STATUS function 5-22, C-4
 storing characters 1-4
 strings
 converting 5-21
 determining length 5-5
 formatting in display positions 5-7
 and multinational characters 5-18
 subroutines 6-3
 subvalue mark 3-11, 3-15
 suppressing zeros 4-14
 system delimiters 3-14, 3-15, 5-25
 in client programs 6-5
 inputting 5-20
 in string variables 5-5
 using SQLGetInfo 6-11, 6-14
 SYSTEM function 5-4, 5-12, C-5

T

tape devices, setting maps for 2-19
 TERM command 2-18, C-3
terminfo file 2-17, 5-12
 TERMINFO function 5-12
 territory 1-7
 definition GL-2
 text mark 3-11, 3-15
 Thai Buddhist Era 4-10
 thousands separators
 specifying in monetary formats 4-16
 specifying in numeric formats 4-14
 Time category 4-3, 4-8
 definition 1-7

TIME command 4-8
 Time records 4-8
 TIMEDATE function 4-8
 TRIM function 5-5
 trim hook function B-21
 type 19 files 4-30, A-4
 T.ATT command C-3

U

UCI 2-21, 6-3
 configuration file 6-9, 6-10
 programs 6-9
 UCI configuration file 6-10
 UNICHAR function 5-20, C-5
 UNICHARS function 5-20, C-5
 Unicode
 block characters, listing 7-3
 block numbers, listing 7-3
 blocks C-10
 definition GL-2
 character set 1-6
 characters 7-3
 listing 7-3
 code point 3-10
 defining characters 3-16
 definition GL-2
 ideographic area 7-3
 menus 7-3
 overview 1-4
 Private Use area 3-16
 replacement character,
 definition GL-3
 shared weights and 4-29
 standard 1-4, 3-16
 values, generating characters from 5-20
 values, generating from characters 5-21
 values, inputting 5-20
 UNICODE keyword 5-24
 UNICODE.FILE command 2-16, 3-18, C-2
 UniNLSLocale object 6-7
 UniNLSMap object 6-7
 UniObjects 2-21, 6-3, 6-6
 UniObjects for Java 2-21, 6-3, 6-7
 UniRPC 6-4, 6-9

UNISEQ function 5-21, C-5
 UNISEQS function 5-21, C-5
 UniVerse Admin Control Panel 2-8
 UniVerse BASIC, *see* BASIC
 UniVerse commands C-2
 UniVerse NLS, *see* NLS
 UniVerse ODBC 2-21, 6-3
 UniVerse Resource service 2-11
 UniVerse servers 6-4
 UniVerse System Administration
 menu 2-4
 UNIX clients 2-22
 UNIX pipes 5-13
 unknown characters
 defining substitute characters for 3-9
 definition GL-3
 unmappable characters
 definition GL-3
 when reading or writing 5-15
 when reading tapes 5-8
 unmapping a file 3-18
 up-arrow mode 5-18
 updating accounts 2-20
 uppercase
 defining characters as 4-20
 rules for converting to lowercase 7-4
 uppercase, defining characters as 4-20
 UPRINT statement 5-13, C-5
usp command 2-15
 UV account directory 1-6, A-1
uvconfig file 2-6, 2-8, 2-11, 2-14, 2-25,
 2-26, 6-9, 6-12, 7-8
 editing 2-8
 UVLANG environment variable 2-13
 UVNLS.H include file 5-4
uvodbc.config file 6-12, 6-13
uvregen command 2-11

calculating 4-31
 shared 4-28
 Windows NT
 multibyte systems 2-22
 WRITE statement 5-5, 5-15

Z

zeros, suppressing in numeric
 formats 4-14

Symbols

!GETPU subroutine 5-13, C-5
 &DEVICE& file 2-14
 &HOLD& file 2-15
 how mapped 2-15
 @ function 5-12

V

value mark 3-11, 3-15
 viewing conventions 4-6

W

weight tables 2-12
 editing 4-30
 weights