Db2 11 for z/OS

Internationalization Guide



SC19-4057-06

Notes

Before using this information and the product it supports, be sure to read the general information under "Notices" at the end of this information.

Subsequent editions of this PDF will not be delivered in IBM Publications Center. Always download the latest edition from PDF format manuals for Db2 11 for z/OS (Db2 Home).

2020-06-17 edition

This edition applies to Db2® 11 for z/OS® (product number 5615-DB2), Db2 11 for z/OS Value Unit Edition (product number 5697-P43), and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© Copyright International Business Machines Corporation 2003, 2020.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this information..... vii
Who should read this information..... vii
Db2 Utilities Suite for z/OS..... vii
Terminology and citations..... viii
Accessibility features for Db2 11 for z/OS..... viii
How to send your comments..... ix
How to read syntax diagrams..... ix

Chapter 1. Introduction to character conversion..... 1
Character conversion terminology..... 1
Code pages and CCSIDs..... 4
Encoding schemes..... 6
ASCII..... 7
EBCDIC..... 8
Unicode..... 10
Endianness..... 14
Situations in which character conversion occurs..... 15
Possible consequences of character conversion..... 16
Types of character conversion..... 16
Expanding conversion..... 16
Contracting conversion..... 18
Round-trip conversion..... 18
Enforced subset conversion..... 19

Chapter 2. How Db2 for z/OS uses Unicode..... 21
Retrieving data from the Db2 catalog..... 22
Specifying that IFCID output should be in Unicode..... 22

Chapter 3. Setting up Db2 to ensure that it interprets characters correctly..... 25
How Db2 performs character conversions..... 25
SYSIBM.SYSSTRINGS catalog table..... 26
Finding the CCSID values of your data sources..... 27
Specifying CCSIDs in Db2..... 29
Specifying subsystem CCSIDs..... 30
Specifying object CCSIDs..... 34
Setting up z/OS Unicode Services for Db2 for z/OS..... 35
Conversion image..... 35
Basic character conversions for Db2 in the z/OS conversion image..... 36
Character conversions for Chinese, Japanese, and Korean character sets in the z/OS conversion image..... 38
Defining additional character conversions..... 40
Checking defined character conversions..... 40

Chapter 4. Storing Unicode data..... 43
Deciding whether to store data as UTF-8 or UTF-16..... 43
Creating a Unicode table..... 44
Tips for handling any extra storage that Unicode data might require..... 46
Estimating the column size for Unicode data..... 47
Inserting data into a Unicode table..... 48
Inserting Unicode data into a non-Unicode table..... 49

Converting existing Db2 data to Unicode....49
Effects on access paths when converting data to Unicode....52

Chapter 5. Application programming with Unicode data and multiple CCSIDs....53
Application encoding scheme....54
Specifying a CCSID for your application....54
Details of CCSID options for application programs....56
Examples of specifying CCSIDs for application data....59
Specifying CCSIDs for COBOL applications when using the Db2 coprocessor....60
Specifying CCSIDs for PL/I applications when using the Db2 coprocessor....63
Specifying CCSIDs for C/C++ applications when using the Db2 coprocessor....66
Determining the CCSID of Db2 data....67
Determining the CCSID of a string value in an SQL statement....68
Objects with different CCSIDs in the same SQL statement....68
Differences between Unicode and EBCDIC sorting sequences....71
Specifying how Db2 calculates the length of a string....73
Specifying the sorting sequence for a language....76
Performing culturally correct case conversions....78
Locale....80
Generating escaped Unicode data....82
Normalization of Unicode strings....84
How Db2 handles Unicode supplementary characters....85
Processing Unicode data in COBOL applications....86
Processing Unicode data in PL/I applications....87
Processing Unicode data in C/C++ applications....87
Java applications and Unicode data....88
Green screen applications and Unicode data....88
Variant characters....89
DRDA character type parameters in Unicode....90

Chapter 6. Debugging CCSID and Unicode problems....91
Potential problems when inserting non-Unicode data into a Unicode table....92

Appendix A. Db2 utilities and Unicode support....93

Appendix B. EXPLAIN Unicode support....95

Appendix C. Db2 ODBC Unicode support....97

Appendix D. IBM Db2 Tools for z/OS Unicode support....99

Appendix E. The International Components for Unicode....101

Appendix F. SYSSTRINGS catalog table....103

Information resources for Db2 11 for z/OS and related products....107

Notices....109
Programming interface information....110
Trademarks....110
Terms and conditions for product documentation....111
Privacy policy considerations....111

Glossary....113

Index..... 115

About this information

This information describes how to handle international data when working in a Db2 11 for z/OS (Db2 for z/OS) environment.

This information provides basic guidance about storing and manipulating Unicode data or data from different code pages in a Db2 for z/OS environment. Topics include detailed information about the following tasks:

1. How to set up your subsystem so that Db2 correctly interprets data in any encoding scheme
2. How to store and manipulate Unicode data
3. How to store and manipulate data in multiple encoding schemes
4. How to write applications that correctly interpret data according to the encoding scheme

Throughout this information, "Db2" means "Db2 11 for z/OS". References to other Db2 products use complete names or specific abbreviations.

Important: To find the most up to date content for Db2 11 for z/OS, always use IBM® Knowledge Center or download the latest PDF file from PDF format manuals for Db2 11 for z/OS (Db2 Home).

This information assumes that Db2 11 is running in new-function mode, and that your application is running with the application compatibility value of 'V11R1'.

Availability of new function in Db2 11

The behavior of data definition statements such as CREATE, ALTER, and DROP, which embed data manipulation SQL statements that contain new capabilities, depends on the application compatibility value that is in effect for the application. An application compatibility value of 'V11R1' must be in effect for applications to use new capability in embedded statements such as SELECT, INSERT, UPDATE, DELETE, MERGE, CALL, and SET *assignment-statement*. Otherwise, an application compatibility value of 'V10R1' can be used for data definition statements.

Generally, new SQL capabilities, including changes to existing language elements, functions, data manipulation statements, and limits, are available only in new-function mode with applications set to an application compatibility value of 'V11R1'.

Optimization and virtual storage enhancements are available in conversion mode unless stated otherwise.

SQL statements can continue to run with the same expected behavior as in DB2® 10 new-function mode with an application compatibility value of 'V10R1'.

Who should read this information

This information is primarily intended for people who are responsible for using character conversion in a Db2 for z/OS environment. It assumes that the user is familiar with the following concepts:

- The basic concepts and facilities of Db2 for z/OS environment
- The basic concepts of Structured Query Language (SQL)

Db2 Utilities Suite for z/OS

Important: In Db2 11, the Db2 Utilities Suite for z/OS is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

Db2 11 utilities can use the DFSORT program regardless of whether you purchased a license for DFSORT on your system. For more information, see the following informational APARs:

- II14047

© Copyright IBM Corp. 2003, 2020

- II14213
- II13495

Db2 utilities can use IBM Db2 Sort for z/OS (5655-W42) as an alternative to DFSORT for utility SORT and MERGE functions. Use of Db2 Sort for z/OS requires the purchase of a Db2 Sort for z/OS license. For more information about Db2 Sort for z/OS, see Db2 Sort for z/OS.

Related concepts
Db2 utilities packaging (Db2 Utilities)

Terminology and citations

When referring to a Db2 product other than Db2 for z/OS, this information uses the product's full name to avoid ambiguity.

About the Db2 brand change: IBM has re-branded DB2 to Db2, and Db2 for z/OS is now the name of the offering previously know as "DB2 for z/OS". During and after implementation of this change, you might sometimes still see references to the original names, such as "DB2 for z/OS" and "DB2", in different IBM web pages and documents. However, assume that these refer to the same "Db2 for z/OS" product when the PID, Entitlement Entity, version, modification, and release information match. For more information, see Revised naming for IBM Db2 family products on IBM z/OS platform.

The following terms are used as indicated:

Db2
Represents either the Db2 licensed program or a particular Db2 subsystem.

Tivoli® OMEGAMON® XE for Db2 Performance Expert on z/OS
Refers to any of the following products:

- IBM Tivoli OMEGAMON XE for Db2 Performance Expert on z/OS
- IBM Db2 Performance Monitor on z/OS
- IBM Db2 Performance Expert for Multiplatforms and Workgroups
- IBM Db2 Buffer Pool Analyzer for z/OS

C, C++, and C language
Represent the C or C++ programming language.

CICS®
Represents CICS Transaction Server for z/OS.

IMS
Represents the IMS Database Manager or IMS Transaction Manager.

MVS™
Represents the MVS element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

RACF®
Represents the functions that are provided by the RACF component of the z/OS Security Server.

Accessibility features for Db2 11 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in z/OS products, including Db2 11 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.

- Customization of display attributes such as color, contrast, and font size

Tip: The IBM Knowledge Center (which includes information for Db2 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard navigation

For information about navigating the Db2 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Related accessibility information

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the commitment that IBM has to accessibility.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other Db2 for z/OS documentation.

Send your comments by email to db2zinfo@us.ibm.com and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).

How to read syntax diagrams

Certain conventions apply to the syntax diagrams that are used in IBM documentation.

Apply the following rules when reading the syntax diagrams that are used in Db2 for z/OS documentation:

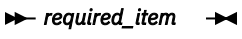
- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►— symbol indicates the beginning of a statement.

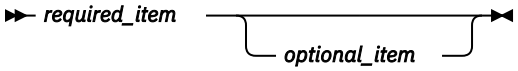
  The —► symbol indicates that the statement syntax is continued on the next line.

  The ►— symbol indicates that a statement is continued from the previous line.

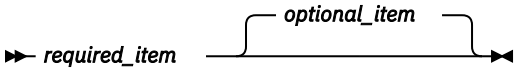
  The —►◄ symbol indicates the end of a statement.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.

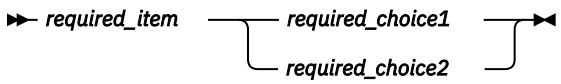
  

  If an optional item appears above the main path, that item has no effect on the execution of the statement and is used only for readability.

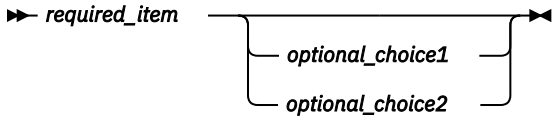
  

- If you can choose from two or more items, they appear vertically, in a stack.

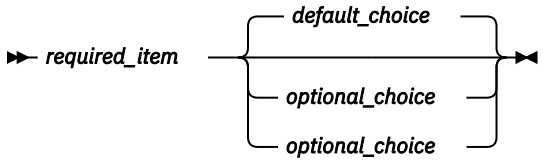
  If you *must* choose one of the items, one item of the stack appears on the main path.



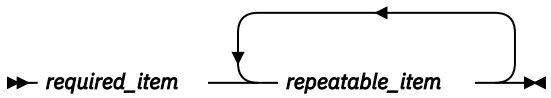
If choosing one of the items is optional, the entire stack appears below the main path.



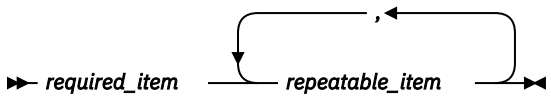
If one of the items is the default, it appears above the main path and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.



If the repeat arrow contains a comma, you must separate repeated items with a comma.

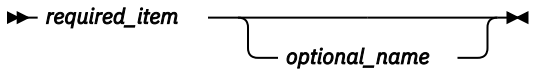


A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.



fragment-name



- With the exception of XPath keywords, keywords appear in uppercase (for example, FROM). Keywords must be spelled exactly as shown. XPath keywords are defined as lowercase names, and must be spelled exactly as shown. Variables appear in all lowercase letters (for example, *column-name*). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

Related concepts

Commands in Db2 (Db2 Commands)
Db2 online utilities (Db2 Utilities)
Db2 stand-alone utilities (Db2 Utilities)

Chapter 1. Introduction to character conversion

In computers, all characters are encoded according to the rules of a particular encoding scheme and code page. If your database and applications handle data from multiple code pages, that data might be converted at certain times from one code page to another. This conversion process is called *character conversion*.

This situation of handling data from multiple code pages is likely if your database and applications contain international data or data from multiple character sets, such as Latin-1 and Katakana. In this situation, character conversions are likely to occur.

The problem with character conversions is that they can degrade performance and potentially cause data loss. Therefore, you should avoid these conversions if possible. One way to avoid these conversions is to have all of your data in one code page. If you use multiple character sets, you might considering using the Unicode code page. This code page includes all characters. If you use Unicode for all of your data, conversions can be avoided. However, converting all of your data to Unicode is not a simple process.

This information discusses basic principles about character conversion and general recommendations that you can apply to your environment for optimal performance and storage.

Character conversion terminology

To understand the concept of character conversion, you should know the meaning of some basic related terms.

The following terms are related to character conversion:

application encoding scheme
The CCSID that your application uses to interpret data in host variables. For Db2 for z/OS applications, typically the application encoding scheme is the value of the ENCODING bind option. (By default the value of the ENCODING bind option is the subsystem default application encoding scheme, which is the APPENSCH DECP value.) However, you can also set the CCSID of application data by using the DECLARE VARIABLE statement with the CCSID option or the CURRENT APPLICATION ENCODING SCHEME special register.

If you are using the Db2 coprocessor, you can use various language compiler options to override the Db2 application encoding scheme for an application.

For more information about application encoding schemes, see “Specifying a CCSID for your application” on page 54

ASCII
Acronym for American Standard Code for Information Interchange, an encoding scheme that is used to represent characters. In this information, the term ASCII is used to refer to IBM-PC data or ISO 8-bit data.

For more information about ASCII, see “ASCII” on page 7. For more information about encoding schemes in general, see “Encoding schemes” on page 6.

big endian
A data format in which the most significant byte is stored first, at the memory location with the lowest address.

For more information about big endian, see “Endianness” on page 14.

character conversion
The process of converting characters from one CCSID to another.

For more information about how Db2 performs character conversions, see “How Db2 performs character conversions” on page 25.

© Copyright IBM Corp. 2003, 2020

character data representation architecture (CDRA)
An IBM architecture that aims to achieve consistent representation, processing, and interchange of graphic character data in data processing environments. CDRA defines a set of identifiers, services, supporting resources, and conventions. The identifiers that CDRA defines are CCSIDs.

For more information about CDRA, see "Code pages and CCSIDs" on page 4.

character repertoire
A set of characters.

character set
A defined set of characters, in which a character is the smallest component of written language that has semantic value.

code page
A specification of code points from a defined encoding scheme for each character in a set or in a collection of character sets. Within a code page, a code point can have only one specific meaning. Code pages are defined by the IBM Globalization Center of Competency.

For more information about code pages, see "Code pages and CCSIDs" on page 4.

code point
A unique bit pattern that represents a character.

For more information about code points, see "Code pages and CCSIDs" on page 4.

coded character set
A set of unambiguous rules that establishes a character set and the one-to-one relationships between the characters of the set and their coded representations. A coded character set is the assignment of each character in a character set to a unique numeric code value.

coded character set identifier (CCSID)
A 16-bit number that identifies a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and additional coding-related information. A CCSID is a number that identifies an implementation of a code page at a particular point in time. A CCSID is an attribute of strings, just as length is an attribute of strings. All values of the same string column have the same CCSID.

For more information about CCSIDs, see "Code pages and CCSIDs" on page 4.

coded character set identifier (CCSID) set
The single byte CCSID value (SBCS), mixed CCSID value, and double byte CCSID value (DBCS) that are associated with a particular encoding scheme.

For more information about CCSID sets, see "Specifying subsystem CCSIDs" on page 30.

collation name
A string value that specifies how Db2 is to sort data. The collation name specifies attributes such as the language of the data, whether case should be considered, and how punctuation characters should be treated.

For more information about collation names, see "Specifying the sorting sequence for a language" on page 76.

contracting conversion
A character conversion in which the length of the converted string is smaller than that of the source string.

For more information about contracting conversions, see "Contracting conversion" on page 18.

conversion image
A data set that contains the information that z/OS Unicode Services needs to perform character and case conversions.

For more information about conversion images, see "Conversion image" on page 35.

EBCDIC
Acronym for Extended Binary-Coded Decimal Interchange Code, a group of coded character sets that consists of 8-bit coded characters. EBCDIC coded character sets assign characters to code points. Each code point consists of 8 bits.

For more information about EBCDIC, see "EBCDIC" on page 8. For more information about encoding schemes in general, see "Encoding schemes" on page 6.

encoding scheme
A set of rules that is used to represent character data. All string data that is stored in a table must use the same encoding scheme. All tables within a table space must use the same encoding scheme, except for global temporary tables, declared temporary tables, and work file table spaces. An encoding scheme only describes the type of encoding; it does not specify code points or a code page. Examples of encoding schemes include ASCII, EBCDIC, and Unicode.

For more information about encoding schemes, see "Encoding schemes" on page 6.

endianness
A data attribute that describes byte order.

For more information about endianness, see "Endianness" on page 14.

enforced subset conversion
A character conversion in which characters that do not have a code point in the target CCSID are converted to a single substitution character.

For more information about enforced subset conversions, see "Enforced subset conversion" on page 19.

escaped data
One or more characters that cannot be represented in the target CCSID and that have been identified as such by some extra syntax.

For more information about escaped data, see "Generating escaped Unicode data" on page 82.

expanding conversion
A character conversion in which the length of the converted string is greater than that of the source string.

For more information about expanding conversions, see "Expanding conversion" on page 16.

International Components for Unicode (ICU)
A set of C/C++ and Java™ libraries for Unicode support and software internationalization.

For more information about ICU, see Appendix E, "The International Components for Unicode," on page 101.

little endian
A data format in which the least significant byte is stored first, at the memory location with the lowest address.

For more information about little endian, see "Endianness" on page 14.

locale
An attribute that defines the user's cultural environment.

For more information about locales, see "Locale" on page 80.

lossless conversion
A character conversion in which all characters in the source CCSID exist in the target CCSID, and thus, no character is lost.

For more information about lossless conversions, see "Possible consequences of character conversion" on page 16.

normalization
A process that produces a unique code point sequence for all sequences that are equivalent, either canonically or compatibly.

For more information about normalization, see “Normalization of Unicode strings” on page 84.

round-trip conversion
A character conversion that ensures the integrity of all character data from the source CCSID to the target CCSID and back to the source. Even if the target CCSID does not support a given character, the character regains its original hexadecimal value after the conversion back to the original CCSID.

For more information about round-trip conversions, see “Round-trip conversion” on page 18.

substitution character
A unique character that is substituted during character conversion for any characters in the source CCSID that do not have a match in the target CCSID.

For more information about substitution characters, see “Enforced subset conversion” on page 19.

supplementary characters
Characters that have a code point between U+10000 and U+10FFFF.

For more information about supplementary characters, see “How Db2 handles Unicode supplementary characters” on page 85.

Unicode
An international character code for information processing that is designed to encode all characters that are used for written communication in a simple and consistent manner. The Unicode character encoding was established to provide enough code points for all the scripts and technical symbols in common usage around the world, plus some ancient scripts.

For more information about Unicode, see “Unicode” on page 10. For more information about encoding schemes in general, see “Encoding schemes” on page 6.

Unicode Consortium
A non-profit organization that develops and maintains international standards, including the Unicode Standard.

For more information about the Unicode Consortium, see Unicode Consortium.

Unicode transformation formats (UTFs)
Forms of Unicode encoding that were devised by the Unicode Consortium to ensure that systems can communicate efficiently. UTF-8, UTF-16, and UTF-32 were each designed for different processing objectives.

For more information about the UTFs, see “UTFs” on page 11.

z/OS Unicode Services
A set of functions that are provided by z/OS. Among the services are case conversion service and character conversion service.

For more information about the z/OS Unicode Services, see “Setting up z/OS Unicode Services for Db2 for z/OS” on page 35.

Code pages and CCSIDs

Because computers store only numbers, they store letters and other characters by assigning a number to them. Which number is mapped to each character depends on the CCSID and code page that is associated with that character.

A *code page* is a mapping of hexadecimal numbers to particular characters. For example, the following table shows code page 37.

Table 1. Code page 37 with CCSID 37

1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(sp)	&	-	ø	Ø	º	µ	^	{	}	\	0

Table 1. Code page 37 with CCSID 37 (continued)

1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-1	(rsp)	é	/	É	a	j	~	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	¢	!	¦	:	⋖	ª	¡	[	(SHY)	¹	²	³
-B	.	$	,	#	⋗	º	¿	]	ô	û	Ô	Û
-C	<	*	%	@	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ú	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	|	¬	?	“	±	¤	®	×	õ	ÿ	Õ	(EO)

Within a code page, each hexadecimal number representation for a character is called a *code point*. When looking at a code page, you can find the hexadecimal code point value for a particular character by concatenating the column header with the row header. For example, find the character 'A' in the preceding code page 37. The character 'A' is in column C and row 1. Therefore, the corresponding code point for the character 'A' is X'C1'. As another example, find the character 'a' in this same code page. The character 'a' is in column 8 and row 1. Therefore, the corresponding code point is X'81'.

A *coded character set identifier* (CCSID) is a number that identifies an implementation of a code page at a particular point in time. For example, the preceding code page 37, which is the US-English code page, has a CCSID of 37.

CCSIDs are defined by the IBM character data representation architecture (CDRA). *CDRA* is an architecture that aims to achieve consistent representation, processing, and interchange of graphic character data in data processing environments. To achieve this consistency, CDRA defines a set of services, supporting resources, conventions, and identifiers, one of which is a CCSID. IBM maintains a repository list of all CCSIDs that are defined by CDRA.

Db2 for z/OS uses CCSIDs. However, Db2 for Linux, UNIX, and Windows uses code pages. The difference between code pages and CCSIDs is the stability. Code pages might change. However, because CCSIDs capture a code page at a particular point in time, the code page that it references does not change.

For example, consider code page 37. At some point, this code page was changed so that code point X'9F' no longer mapped to the international currency symbol (¤). Instead, this code point was mapped to the euro symbol (€). CCSID 37 refers to the original code page 37. The altered code page has CCSID 1140. CCSID 1140 and CCSID 37 differ by only that one character at code point X'9F'. The following table shows CCSID 1140.

Table 2. Code page 37 with CCSID 1140

1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(sp)	&	-	ø	Ø	º	µ	^	{	}	\	0
-1	(rsp)	é	/	É	a	j	~	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	¢	!	¦	:	⩽	ª	¡	[	(SHY)	¹	²	³
-B	.	$	,	#	⩾	º	¿	]	ô	û	Ô	Û
-C	<	*	%	@	ð	æ	Ð	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ú	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	|	¬	?	“	±	€	®	×	õ	ÿ	Õ	(EO)

The exception to this idea of fixed CCSIDs is the CCSID set that Db2 for z/OS uses for Unicode code pages. For Unicode data, Db2 for z/OS uses CCSIDs that have the ability to grow as the Unicode standard grows. For more information about those CCSIDs, see “Unicode CCSIDs” on page 13.

In Db2 for z/OS, all character data is associated with a CCSID. If the data does not have one, Db2 uses the subsystem defaults. You specify these subsystem default CCSID values when you install Db2. Character conversion is described in terms of CCSIDs of the source and target.

Related concepts

Euro symbol support (Db2 Installation and Migration)

Related tasks

Specifying CCSIDs in Db2

You must communicate to Db2 the correct CCSIDs to use for your data to ensure that Db2 correctly interprets your data. You can specify default subsystem CCSIDs. You can also specify CCSIDs for individual applications and Db2 objects.

Related information

Coded character set identifiers (CCSIDs)

Encoding schemes

An *encoding scheme* standardizes the encoding of character sets by defining a set of rules for representing character data. Each encoding scheme consists of a number of code pages that adhere to its rules. For example, code pages 37, 500, and 1047 are all part of the EBCDIC encoding scheme.

The major encoding schemes are EBCDIC, ASCII, and Unicode. The EBCDIC encoding scheme is typically used on IBM Z (z/OS) and iSeries (AS/400). The ASCII encoding scheme is used on Intel-based systems,

such as Windows, UNIX based systems, such as AIX®, and the Linux operating system. The Unicode encoding scheme is supported by many operating systems.

ASCII

The American Standard Code for Information Interchange (ASCII) is an encoding scheme. ASCII is typically used on Intel-based systems, such as Windows, and UNIX-based systems, such as Linux.

ASCII was developed by a committee of the American Standards Association. The first ASCII standard was published in 1963.

Certain characters are the same on every ASCII code page. Those characters are called *invariant characters*. Other characters might vary from one code page to the next. These characters are called *variant characters*. For more information about ASCII invariance in SBCS code pages, see Invariance of the Syntactic Character Set in Basic SBCS Encoding Structures .

Some example ASCII CCSIDs are 367, 819, and 912.

The following table shows the code page for ASCII CCSID 367.

Table 3. CCSID 367

1st → / 2nd ↓	0-	1-	2-	3-	4-	5-	6-	7-
-0			(sp)	0	@	P	`	p
-1			!	1	A	Q	a	q
-2			"	2	B	R	b	r
-3			#	3	C	S	c	s
-4			$	4	D	T	d	t
-5			%	5	E	U	e	u
-6			&	6	F	V	f	v
-7			'	7	G	W	g	w
-8			(	8	H	X	h	x
-9			)	9	I	Y	i	y
-A			*	:	J	Z	j	z
-B			+	;	K	[	k	{
-C			,	<	L	\	l	|
-D			-	=	M	]	m	}
-E			.	>	N	^	n	~
-F			/	?	O	_	o	

Related concepts

Variant characters

Variant characters are characters that correspond to different code points across a given set of code pages. For example, the character # is variant. It corresponds to code point X'7B' in CCSIDs 37, 273, 500, and 1047. However, this character corresponds to code point X'4A' in CCSID 277.

EBCDIC

Extended Binary Coded Decimal Interchange Code (EBCDIC) is an encoding scheme that is typically used on IBM Z (z/OS) and iSeries (System i).

EBCDIC was developed by IBM in 1963.

Certain characters are the same on every EBCDIC code page. Those characters are called *invariant characters*. Other characters might vary from one code page to the next. These characters are called *variant characters*. For more information about EBCDIC invariance in SBCS code pages, see Invariance of the Syntactic Character Set in Basic SBCS Encoding Structures .

Some example EBCDIC CCSIDs are 37, 500, and 1047.

The following table shows the code page for EBCDIC CCSID 37.

Table 4. Code page 37 with CCSID 37

1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(sp)	&	-	ø	Ø	º	µ	^	{	}	\	0
-1	(rsp)	é	/	É	a	j	~	£	A	J	÷	1
-2	â	ê	Â	Ê	b	k	s	¥	B	K	S	2
-3	ä	ë	Ä	Ë	c	l	t	•	C	L	T	3
-4	à	è	À	È	d	m	u	©	D	M	U	4
-5	á	í	Á	Í	e	n	v	§	E	N	V	5
-6	ã	î	Ã	Î	f	o	w	¶	F	O	W	6
-7	å	ï	Å	Ï	g	p	x	¼	G	P	X	7
-8	ç	ì	Ç	Ì	h	q	y	½	H	Q	Y	8
-9	ñ	ß	Ñ	`	i	r	z	¾	I	R	Z	9
-A	¢	!	¦	:	⩽	ª	¡	[	(SHY)	¹	²	³
-B	.	$	,	#	⩾	º	¿	]	ô	û	Ô	Û
-C	<	*	%	@	ð	æ	Đ	¯	ö	ü	Ö	Ü
-D	(	)	_	'	ý	¸	Ý	¨	ò	ú	Ò	Ù
-E	+	;	>	=	þ	Æ	Þ	´	ó	ú	Ó	Ú
-F	|	¬	?	“	±	¤	®	×	õ	ÿ	Õ	(EO)

Related concepts

Variant characters

Variant characters are characters that correspond to different code points across a given set of code pages. For example, the character # is variant. It corresponds to code point X'7B' in CCSIDs 37, 273, 500, and 1047. However, this character corresponds to code point X'4A' in CCSID 277.

Code point differences between EBCDIC CCSIDs

Although many EBCDIC code pages are similar, code points for certain characters vary from code page to code page. These characters are called variant characters and can potentially cause problems.

Characters A-Z, a-z, and 0-9 correspond to the same hexadecimal code points on most EBCDIC code pages. Other characters, such as the left bracket ([) correspond to different code points depending on the CCSID. Therefore, to ensure that Db2 interprets your data correctly, you should specify the correct CCSID, especially when you use characters other than A-Z, a-z, and 0-9.

The following tables show some code point differences between several common EBCDIC CCSIDs.

The following table shows the difference in code points between EBCDIC CCSID 37 and EBCDIC CCSID 500.

Table 5. Code point differences between EBCDIC CCSID 37 and EBCDIC CCSID 500

Code point	Character	
	CCSID 37	**CCSID 500**
X'4A'	¢ (cent sign)	[(left bracket)
X'4F'	| (vertical bar)	! (exclamation point)
X'5A'	! (exclamation point)	] (right bracket)
X'5F'	¬(logical not)	^ (circumflex accent)
X'B0'	^(circumflex accent)	¢ (cent sign)
X'BA'	[(left bracket)	¬(logical not)
X'BB'	] (right bracket)	| (vertical bar)

The following table shows the difference in code points between EBCDIC CCSID 37 and EBCDIC CCSID 1047.

Table 6. Code point differences between EBCDIC CCSID 37 and EBCDIC CCSID 1047

Code point	Character	
	CCSID 37	**CCSID 1047**
X'5F'	¬(logical not)	^(circumflex accent)
X'AD'	Ý (uppercase Y with acute accent)	[(left bracket)
X'B0'	^(circumflex accent)	¬(logical not)
X'BA'	[(left bracket)	Ý (uppercase Y with acute accent)
X'BB'	] (right bracket)	¨ (umlaut)
X'BD'	¨ (umlaut)	] (right bracket)

The following table shows the difference in code points between EBCDIC CCSID 500 and EBCDIC CCSID 1047.

Table 7. Code point differences between EBCDIC CCSID 500 and EBCDIC CCSID 1047

Code point	Character	
	CCSID 500	**CCSID 1047**
X'4A'	[(left bracket)	¢(cent sign)
X'4F'	! (exclamation point)	| (vertical bar)
X'5A'	] (right bracket)	! (exclamation point)
X'AD'	Ý (uppercase Y with acute accent)	[(left bracket)
X'B0'	¢(cent sign)	¬(logical not)
X'BA'	¬(logical not)	Ý (uppercase Y with acute accent)
X'BB'	| (vertical bar)	¨ (umlaut)
X'BD'	¨ (umlaut)	] (right bracket)

Related concepts

Variant characters

Variant characters are characters that correspond to different code points across a given set of code pages. For example, the character # is variant. It corresponds to code point X'7B' in CCSIDs 37, 273, 500, and 1047. However, this character corresponds to code point X'4A' in CCSID 277.

Unicode

Unicode is an encoding scheme that currently provides a unique code point for over 100,000 characters. This standard enables systems to more easily handle global data, regardless of the platform, program, or language.

Before Unicode was defined, no single encoding was adequate for all available letters and symbols. For example, consider the following restrictions for EBCDIC and ASCII:

- These encoding schemes have one code page per character set. For example, they have one code page for Japanese characters and another code page for German characters.
- These encoding schemes often encode data in different positions. For example, the letter A is encoded as X'C1' in most EBCDIC code pages, but it is encoded as X'41' in most ASCII code pages.
- Even within encoding schemes, characters might be mapped differently. For example, the letter ä is encoded as X'C0' in EBCDIC code page 273, but it is encoded as X'43' in EBCDIC code page 37. (Code page 37 has the left brace character ({) at position X'C0'.) This same letter ä is encoded as X'E4' in ASCII code page 819 and as X'7B' in ASCII code page 1011.

Thus, handling data from more than one character set, such as German characters and Arabic characters, was difficult when ASCII or EBCDIC was used.

Unicode avoids these problems by having a single standard that can provide a unique code point for over a million characters. Currently, the standard has defined code points for just over 100,000 characters. You can view the Unicode code points by looking at the Unicode character code charts on the Unicode Consortium web site. For example, if you look up Unicode code point U+41, you can see that it corresponds to the character 'A'.

The following table shows the first 128 Unicode code points from U+00 to U+7E. These code points are the same as those in ASCII 367.

Table 8. The first 128 code points for Unicode and ASCII CCSID 367

1st → 2nd ↓	0-	1-	2-	3-	4-	5-	6-	7-
	NUL	DLE	(sp)	0	@	P	`	p
	SCH	DC1	!	1	A	Q	a	q
	STX	DC2	"	2	B	R	b	r
	ETX	DC3	#	3	C	S	c	s
	EQT	DC4	$	4	D	T	d	t
	ENQ	NAK	%	5	E	U	e	u
	ACK	SYN	&	6	F	V	f	v
	BEL	ETB	'	7	G	W	g	w
	BS	CAN	(	8	H	X	h	x
	HT	EM	)	9	I	Y	i	y
	LF	SUB	*	:	J	Z	j	z
	VT	ESC	+	;	K	[	k	{
	FF	FS	,	<	L	\	l	|
	OR	GS	-	=	M	]	m	}
	SO	RS	.	>	N	^	n	~
	SI	US	/	?	O	_	o	DEL

Related concepts

Code pages and CCSIDs

Because computers store only numbers, they store letters and other characters by assigning a number to them. Which number is mapped to each character depends on the CCSID and code page that is associated with that character.

Related reference

z/OS Displaying Unicode Services (z/OS MVS System Commands)

Related information

Unicode Consortium

Unicode Character Code Charts (on Unicode Consortium website)

UTFs

Each Unicode code point can be expressed in several different formats. These formats are called *Unicode transformation formats* (UTFs). For example, the letter M is the Unicode code point U+004D. In UTF-8, this code point is represented as X'4D'. In UTF-16, this code point can be represented as X'004D'. [1]

A UTF maps each Unicode code point to a unique code unit sequence. A *code unit* is the minimal bit combination that can represent a character. Each UTF uses a different code unit size. For example, UTF-8 is based on 8-bit code units. Therefore, each character can be 8 bits (1 byte), 16 bits (2 bytes), 24 bits (3 bytes), or 32 bits (4 bytes). Likewise, UTF-16 is based on 16-bit code units. Therefore, each character can be 16 bits (2 bytes) or 32 bits (4 bytes).

[1] X'004D' is the UTF-16 big endian representation. The UTF-16 little endian representation is X'4D00'. For more information about endianness, see “Endianness” on page 14.

All UTFs include the full Unicode *character repertoire*, or set of characters. Each UTF can represent any Unicode character that you need to represent.

The following UTFs are defined by the Unicode Consortium:

UTF-8

UTF-8 is based on 8-bit code units. Each character is encoded as 1 to 4 bytes.

The first 128 Unicode code points are encoded as 1 byte in UTF-8. These code points are the same as those in ASCII CCSID 367. Any other character is encoded with more than 1 byte in UTF-8.

In IBM, UTF-8 is also known as Unicode CCSID 1208.

Db2 uses UTF-8 to encode data in the following ways:

- Db2 uses UTF-8 to encode data in CHAR, VARCHAR, and CLOB columns in Unicode tables.
- Db2 parses SQL statements and precompiles source code in UTF-8.
- The Db2 catalog tables that have the Unicode encoding scheme are encoded in UTF-8.

UTF-16

UTF-16 is based on 16-bit code units. Each character is encoded as at least 2 bytes. Some characters that are encoded with a 1-byte code unit in UTF-8 are encoded with a 2-byte code unit in UTF-16.

Characters that are surrogate or supplementary characters use 4 bytes and thus require additional storage. These characters can also be stored in UTF-8 or UTF-32, but, because they always require 4 bytes of storage, neither of these formats provide any space savings.

In IBM, UTF-16 is also known as Unicode CCSID 1200.

Db2 uses UTF-16 to encode data in GRAPHIC, VARGRAPHIC, and DBCLOB columns in Unicode tables.

UTF-32

UTF-32 is based on 32-bit code units. Each character is encoded as at least 4 bytes. Db2 does not store data in UTF-32.

The following table shows example UTF encodings for several characters.

Table 9. Example UTF encodings

Character	Unicode code point	ASCII	UTF-8	UTF-16 (Big Endian format)[1]	UTF-32 (Big Endian format)
A	U+0041	X'41'	X'41'	X'0041'	X'00000041'
a	U+0061	X'61'	X'61'	X'0061'	X'00000061'
9	U+0039	X'39'	X'39'	X'0039'	X'00000039'
Å	U+00C5	X'C5'	X'C385'[2]	X'00C5'	X'000000C5'
	U+9860	X'CDDB' (CCSID 939)	X'E9A1A0'	X'9860'	X'00009860'
𠀐	U + 200D0	Does not exist	X'F0A08390'	X'D840DCD0'	X'000200D0'

Notes:

1. z/OS uses Big Endian format only. Little Endian format is used in other operating systems.
2. X'C5' becomes double-byte in UTF-8.

Notice that for some characters, the UTF encodings are fairly predictable. For example, the character A, which is Unicode code point U+0041, is encoded as X'41' in ASCII and UTF-8, and as X'0041' in UTF-16 and as X'00000041' in UTF-32. However, the UTF encodings for a character like Å or𠀐 do not follow the same pattern.

The process of converting a value from its Unicode code point to its UTF hexadecimal value is called *encoding*. For example, Unicode code point U+0041 is encoded in UTF-8 as X'41'. The reverse process, converting a UTF hexadecimal value to its Unicode code point, is called *decoding*. For example, suppose that you see the hexadecimal value X'00C5' in trace output and you know that the data is in UTF-16. You can decode the value to find that it corresponds to Unicode code point U+00C5. You can then look up this Unicode code point on the Unicode character code charts on the Unicode Consortium web site and find that it corresponds to the character Å.

You can find the steps for how to manually encode and decode Unicode data on the Unicode Consortium web site. Alternatively, you can use a converter tool to do the conversion for you.

Related concepts

Endianness

Endianness is a data attribute that describes byte order. When applications exchange data, they need to know the ordering convention for multi-byte data. Otherwise, data can be misinterpreted.

Related information

Unicode Consortium

UTF-8, UTF-16, UTF-32 & BOM (on Unicode Consortium website)

Unicode Character Code Charts (on Unicode Consortium website)

CCSID 367 code page

Unicode CCSIDs

Db2 for z/OS uses CCSIDs 367, 1200, and 1208 for Unicode data.

367

Db2 uses ASCII CCSID 367 for single-byte character data (SBCS) because the first 128 code points in Unicode UTF-8 are the same as the those in ASCII CCSID 367.

Therefore, Db2 uses CCSID 367 for CHAR, VARCHAR, and CLOB columns that are defined with FOR SBCS DATA in Unicode tables.

1208

Db2 uses CCSID 1208 for Unicode UTF-8 data, which Db2 always considers to be mixed data. This CCSID is the default CCSID value for Unicode tables.

Therefore, Db2 uses CCSID 1208 for CHAR, VARCHAR, and CLOB columns that are defined with FOR MIXED DATA in Unicode tables. FOR MIXED DATA is the default subtype specification.

1200

Db2 uses CCSID 1200 for Unicode UTF-16 data, which is double-byte data (DBCS). This CCSID applies to GRAPHIC and VARGRAPHIC Unicode data.

Therefore, Db2 uses CCSID 1200 for GRAPHIC, VARGRAPHIC, and DBCLOB columns in Unicode tables.

CCSIDs usually refer to a code page at a particular point in time. However, the Unicode CCSIDs that Db2 for z/OS uses are an exception. They can expand to include more characters as the Unicode standard grows. For example, CCSID 1200 can include the characters from the Unicode standard code pages 13488 (Unicode 2.0) and 17584 (Unicode 3.0). You can determine the CCSID for each Unicode standard code page by looking at the list of registered CCSIDs.

Because Db2 uses this architecture for CCSIDs, you can easily migrate to new versions of the Unicode standard by just updating your conversion image. However, the disadvantage to this architecture is that the CCSID value does not clearly tell you which characters are supported. To check which Unicode standard is currently supported for a particular conversion, issue the system DISPLAY UNI command.

Example DISPLAY UNI command: The following example output is from the command `d uni,all`:

```
CUN3000I 09.33.37 UNI DISPLAY 754
 ENVIRONMENT: CREATED       01/25/2010 AT 00.20.12
              MODIFIED      01/25/2010 AT 00.25.10
              IMAGE CREATED --/--/---- AT --.--.--
     SERVICE: CHARACTER      CASE           NORMALIZATION  COLLATION
              STRINGPREP     BIDI           CONVERSION INF
```

```
       STORAGE: ACTIVE       1995 PAGES
                FIXED           0 PAGES
                LIMIT      524288 PAGES
      CASECONV: ENABLED
      CASE VER: UNI300  NORMAL  SPECIAL LOCALE
     NORMALIZE: DISABLED
      NORM VER: NONE
       COLLATE: DISABLED
    COLL RULES: NONE
   STRPROFILES: NONE
    CONVERSION: 00367-05123-R               00437-00819-R
                00273-01208-R               01140-01252-E
                01140-01252-R               00437-00850-E
                00437-00850-R               01200(17584)-01140-E
                01200(17584)-01141-E        01200(17584)-01142-E
                01200(17584)-01144-E        00273-01252-E
                00273-01252-R               01200(17584)-01148-E
                00367-05210-R               00850-01200(13488)-R
                01142-00367-E               00836-00367-E
                01386-00836-R               01148-01200(17584)-R
                01386-00935-RE              00437-01140-E
                00437-01140-R               00437-01148-E
                00437-01148-R               00437-01208-R
     ...
```

The CONVERSION section of this output lists all of the CCSID conversions that are defined. For example, the line `01200(17584)-01141-E` defines the conversion between CCSID 1200 and CCSID 1141. Db2 uses CCSID 1200 for Unicode UTF-16 data. The number in parentheses after CCSID 1200, 17584, means that in this conversion, CCSID 1200 uses Unicode standard 3.0. In the line `00850-01200(13488)-R`, CCSID 1200 is followed by a different number, 13488. For this conversion, CCSID 1200 uses Unicode standard 2.0. The letters E and R represent the type of conversion. E means that the conversion is an enforced subset conversion. R means that the conversion is a round-trip conversion.

Endianness

Endianness is a data attribute that describes byte order. When applications exchange data, they need to know the ordering convention for multi-byte data. Otherwise, data can be misinterpreted.

Data can have the following byte order formats:

Big endian
A format in which the most significant byte is stored first. The other bytes follow in decreasing order of significance. For example, for a four-byte word, the byte order is 0, 1, 2, 3. For a two-byte word, it is 0, 1.

Big endian format is used by pSeries, IBM Z, iSeries, Sun, and HP.

Little endian
A format in which the least significant byte is stored first. The other bytes follow in increasing order of significance. For example, for a four-byte word, the byte order is 3, 2, 1, 0. For a two-byte word, it is 1, 0.

Little endian format is used by Intel-based machines, including xSeries.

Endianness affects only multi-byte data. Within a single byte, the bits are always ordered in the same way. Bit order within a byte is always 7, 6, 5, 4, 3, 2, 1, 0.

UTF-8 data is not affected by endianness, even if the data is stored as more than 1 byte. UTF-16 data and UTF-32 data are affected by endianness. For example, the character 'A' is encoded for UTF-16 and UTF-32 as shown in the following table:

Table 10. Example encoding for the character 'A'

	UTF-16	**UTF-32**[1]
Big endian	X'0041'	X'00000041'
Little endian	X'4100'	X'41000000'

Table 10. Example encoding for the character 'A' (continued)

	UTF-16	UTF-32[1]

Note:

1. **Db2 for z/OS does not store data in UTF-32**

Endianness becomes a potential problem when data is exchanged between systems and applications that use different endian formats and the data is not properly converted. Be aware of the endian format of the data that your system or application handles. You might notice endianness problems when looking at character encoding values in traces. Such a problem might exist if you notice that numeric byte values have been switched. For example, you expect X'0041' but see X'4100'.

Example: Suppose that you are loading data in UTF-16 little endian format (CCSID 1202) from a .NET application. Db2 for z/OS does not support storing data in CCSID 1202. However, Db2 does support conversions to and from CCSID 1202. Thus, Db2 converts the data and stores it in UTF-16 big endian format (CCSID 1200). In this case, you should be aware that the data format has changed.

Related reference

UTFs

Situations in which character conversion occurs

Character conversion is the process of converting data from one CCSID to another CCSID. This process can occur when data is transferred between a remote and local system or when data is manipulated within the local system.

Character conversion is more likely to occur when you are accessing data remotely because this situation often involves different platforms and encoding schemes. For example, in a client/server environment, a requester might send the values of host variables in SELECT predicates and INSERT column values to the current server. The current server might then send the values of result columns back to the requester. In either transaction, if the string data has a different representation at the sending and receiving systems, conversion occurs.

Conversion can also occur during string operations on the same system, as in the following examples:

- A DECLARE VARIABLE statement specifies an overriding CCSID.
- The SQLDA specifies an overriding CCSID for a string column.
- You compare or combine data from multiple CCSIDs in an SQL statement.
- You use SPUFI, which processes EBCDIC data, to insert data into a Unicode table.
- The value of the ENCODING bind option (for static SQL statements) or the CURRENT APPLICATION ENCODING SCHEME special register (for dynamic SQL statements) is different than the encoding scheme of the data that is being inserted or retrieved.
- An ASCII or EBCDIC application provides SQL statement text to Db2 in a PREPARE statement. Db2 converts the statement text to Unicode for parsing.

Related concepts

Objects with different CCSIDs in the same SQL statement

You can reference data with different CCSIDs from the same SQL statement. This ability is useful if you use table objects such as tables, views, temporary tables, query tables, and user-defined functions with different CCSIDs. However, you should understand how Db2 for z/OS processes these queries so that you can code them correctly.

Related reference

DECLARE VARIABLE (Db2 SQL)

SQL descriptor area (SQLDA) (Db2 SQL)

BIND and REBIND options for packages, plans, and services (Db2 Commands)

CURRENT APPLICATION ENCODING SCHEME (Db2 SQL)

PREPARE (Db2 SQL)

Possible consequences of character conversion

You should try to avoid character conversions when possible, because conversions can potentially slow performance and sometimes cause data loss. The way to avoid conversions is to use the same CCSID for all of your data.

The best character conversion is no conversion, because conversion always has a performance cost. The cost depends on the extent of the conversion. For example, if you have a Unicode table and select every row into an EBCDIC application, the performance cost is probably noticeable. However if you issue a `SELECT MAX(xxxx) FROM` on a Unicode table, and then convert the result to EBCDIC, you might not notice the performance cost.

The second best conversion is a lossless conversion. A *lossless conversion* is one in which all characters in the source CCSID exist in the target CCSID and thus, no character is lost. For example, a conversion from CCSID 37 to CCSID 500 is lossless, because they both include the same set of characters. The difference between these two CCSIDs is the placement of 7 characters. These seven characters have different code points in each of these CCSIDs. A conversion from CCSID 1208 (UTF-8) to CCSID 1200 (UTF-16) is also lossless, because they both include the same repertoire of characters.

If the conversion is not a lossless conversion, certain characters might be lost. ("Lost" means that these characters are replaced by substitution characters.) Thus, the integrity of your data can be compromised.

Types of character conversion

Character conversions can be characterized by their effect on the length of the string. Conversions can be expanding, contracting or neither. Character conversions can also be characterized by how they handle characters that do not exist in the target CCSID. They can be round-trip conversions or enforced subset conversions.

Expanding conversion

An *expanding conversion* occurs when the length of the converted string is greater than that of the source string.

For example, an expanding conversion occurs when an ASCII mixed data string that contains DBCS characters is converted to an EBCDIC mixed data string. The expansion occurs because of the addition of shift-out and shift-in control characters. Expanding conversions can also occur when string data is converted to or from Unicode.

Expanding conversions typically affect European and Asian Pacific languages. For example, the German name Jürgen expands when it is converted from ASCII or EBCDIC to Unicode. Also, Japanese, Korean, and Chinese strings expand when they are converted from ASCII to EBCDIC.

Expanding conversions can have the following effects on Db2:

- Expanding conversions might cause problems with fixed-length variables. For example, when a fixed-length host variable needs to be converted from ASCII mixed data to EBCDIC mixed data, an error occurs. The problem occurs because the conversion is an expanding conversion, but the host variable is fixed-length. The solution is to use a varying-length string variable with a maximum length that is sufficient to contain the expansion.
- Expanding conversions can affect fixed-length strings. If you use a fixed-length string and an expanding conversion occurs, Db2 truncates the string. Db2 examines the characters that are being truncated to ensure that significant data is not truncated. For example, trailing blanks are insignificant. In this situation, consider using the VARCHAR data type for these strings.
- Expanding conversions can affect the results of length functions, such as LENGTH, CHARACTER_LENGTH, SUBSTRING, and SUBSTR on the converted string. For CHARACTER_LENGTH and SUBSTRING, use the CODEUNITS16 and CODEUNITS32 options to limit the effects of expanding conversions.

- Expanding conversions can affect the length of the object names, such as table names and column names. You can avoid these problems by not using special characters in object names.

To determine the worst-case result length of a CCSID conversion, use the following table.

Table 11. Worst case result length of CCSID conversion, where X represents LENGTH(string in bytes)

		To CCSID								
		EBCDIC			ASCII			Unicode		
From CCSID		**SBCS**	**Mixed**	**DBCS**	**SBCS**	**Mixed**	**DBCS**	**SBCS**	**UTF-8**	**UTF-16**
EBCDIC	SBCS	X	X	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	Mixed	X	X	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	DBCS	X*0.5[1]	X+2	X	X*0.5[1]	X	X	X*0.5	X*1.5	X
ASCII	SBCS	X	X	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	Mixed	X	X*1.8	X*2[1]	X	X	X*2[1]	X[1]	X*3	X*2
	DBCS	X*0.5[1]	X+2	X	X*0.5[1]	X	X	X*0.5	X*1.5	X
Unicode	SBCS	X	X	X*2	X	X	X*2	X	X	X*2
	UTF-8	X	X*1.25	X	X	X	X	X	X	X*2
	UTF-16	X*0.5	X+2	X	X*0.5	X	X	X*0.5	X*1.5	X

Note:

1. Because of the high probability of data loss, IBM does not provide conversion tables for this combination of two CCSIDs and data subtypes.

Example: In ASCII CCSID 819, the character Å is represented by the code point X'C5'. In UTF-8 CCSID 1208, this character is represented by X'C385'. Thus, the conversion of the character Å from CCSID 819 to CCSID 1208 is an expanding conversion.

Example: The following table shows a string with Kanji characters and Latin characters in different encoding schemes.

Table 12. Example of a character string in different encoding schemes

Data type and encoding scheme	Character representation	Hexadecimal representation (with spaces separating each character)
9 bytes in ASCII	元 gen 気 ki	8CB3 67 65 6E 8B43 6B 69
13 bytes in EBCDIC	SO 元 SI gen SO 気 SI ki	0E 4695 0F 87 85 95 0E 45B9 0F 92 89
11 bytes in Unicode UTF-8	元 gen 気 ki	E58583 67 65 6E E6B097 6B 69

If you convert this string from ASCII to EBCDIC, notice that shift-in and shift-out characters are added. This conversion is an example of an expanding conversion. The length increases from 9 bytes to 13 bytes.

Related reference

CHARACTER_LENGTH (Db2 SQL)
LENGTH (Db2 SQL)
SUBSTR (Db2 SQL)
SUBSTRING (Db2 SQL)

Contracting conversion

A *contracting conversion* occurs when the length of the converted string is smaller than that of the source string.

For example, a contracting conversion occurs when an EBCDIC mixed data string that contains DBCS characters is converted to ASCII mixed. The contraction occurs because shift characters are removed. Contracting conversions can also occur when string data is converted to or from Unicode.

Contracting conversions typically affect European and Asian Pacific languages. For example, the German word straße contracts when it is converted from Unicode to ASCII or EBCDIC. Also, Japanese, Korean, and Chinese strings might contract when they are converted from EBCDIC to ASCII.

Contracting conversions can have the following effects on Db2:

- Contracting conversions can affect the results of the length functions, such as LENGTH, CHARACTER_LENGTH, SUBSTRING, and SUBSTR, on the converted string. For CHARACTER_LENGTH and SUBSTRING, use the CODEUNITS16 and CODEUNITS32 options to limit the effects of contracting conversions.
- Contracting conversions can affect the length of the object names, such as table names and column names. You can avoid these problems by not using special characters in object names.

To determine the worst-case result length of a CCSID conversion, use the table in “Expanding conversion” on page 16.

Example: In UTF-8 CCSID 1208, the character Å is represented by the code point X'C385'. In ASCII CCSID 819, this character is represented by X'C5'. Thus, the conversion of the character Å from CCSID 1208 to CCSID 819 is a contracting conversion.

Example: The German name Jürgen contracts when it is converted from Unicode to ASCII or EBCDIC and expands when it is converted from ASCII or EBCDIC to Unicode.

Related concepts

String unit specifications (Db2 SQL)

Related reference

CHARACTER_LENGTH (Db2 SQL)
LENGTH (Db2 SQL)
SUBSTR (Db2 SQL)
SUBSTRING (Db2 SQL)

Round-trip conversion

A *round-trip conversion* ensures the integrity of all character data from the source CCSID to the target CCSID and back to the source. Even if the target CCSID does not support a given character, the character regains its original hexadecimal value after it is converted back to the source CCSID.

One alternative to a round-trip conversion is an enforced subset conversion, during which characters that do not exist in the target CCSID are lost. Whether a particular conversion uses a round-trip conversion or an enforced subset conversion depends on how your system is set up to do conversions. For example, in Db2 for z/OS, many conversions are defined by z/OS Unicode Services. Each of the conversion definitions specifies whether to use a round-trip or enforced subset conversion.

A round-trip conversion works only in a two-tier homogenous environment where the data makes the complete round trip. For example, if you pass data from Db2 for Linux, UNIX, and Windows to Db2 for z/OS and then back to Db2 for Linux, UNIX, and Windows with a round-trip conversion, no data is lost. The data was converted back to its original format. However, if you have a more complicated environment, a round-trip conversion does not necessarily preserve data integrity. For example, if you pass data from Db2 for z/OS to Db2 for Linux, UNIX, and Windows and then to Linux on a Java client, two conversions have potentially occurred. Because the data was not converted back to its original format before the second conversion, data might have been lost even if round-trip conversions are used.

Example: In ASCII CCSID 1252, the trademark symbol ™ is code point X'99'. In EBCDIC CCSID 37, this code point does not exist. During character conversion from ASCII CCSID 1252 to EBCDIC CCSID 37, the trademark symbol is converted to a control character, X'39'. If you use SPUFI to select the data, the data displays as the unprintable character that you specified in your emulator, which is generally a quotation mark (") or a period (.). If you issue a SELECT HEX(*column*) statement, the data displays as X'39', which is the DEL control character. When the client uses a round-trip conversion to convert this character back to ASCII CCSID 1252, the trademark symbol is preserved as code point X'99'. Notice that this conversion is not lossless unless it is converted back to the original format.

Enforced subset conversion

An *enforced subset conversion* occurs when a character in the source CCSID does not have a code point in the target CCSID. In this case, the character is converted to a single substitution character.

The default substitution characters (SUB) are:

- X'3F' for SBCS EBCDIC
- X'1A' or X'7F' for SBCS ASCII
- X'1A'for UTF-8
- X'001A' for UTF-16

For DBCS data, the substitution character varies depending on the CCSID.

One alternative to an enforced subset conversion is a round-trip conversion, which preserves characters if they are converted back to the originally CCSID. Whether a particular conversion uses a round-trip conversion or an enforced subset conversion depends on how your system is set up to do conversions. For example, in Db2 for z/OS, many conversions are defined by z/OS Unicode Services. Each of the conversion definitions specifies whether to use a round-trip or enforced subset conversion.

Example: In ASCII CCSID 1252, the trademark symbol ™ is represented by the code point X'99'. In EBCDIC CCSID 37, this code point does not exist. During an enforced subset character conversion to EBCDIC CCSID 37, this code point is converted to the substitution character X'3F'. When the code point is converted back to ASCII CCSID 1252, the character remains a substitution character and is represented by the code point X'1A'.

Example: In ASCII CCSID 5348, the euro symbol (€) is represented by the code point X'80'. In EBCDIC CCSID 37, this code point does not exist. During an enforced subset character conversion to EBCDIC CCSID 37, this code point is converted to the substitution character X'3F'. When the code point is converted back to ASCII CCSID 5348, the character remains a substitution character and is represented by the code point X'1A'.

z/OS Unicode Services uses enforced subset conversions when converting from Unicode to ASCII or EBCDIC to handle characters that do not exist in the target CCSID. In this situation, enforced subset conversions are required because Unicode has room to include over 1 million code points, but ASCII and EBCDIC single-byte character sets can include only 256 code points.

Chapter 2. How Db2 for z/OS uses Unicode

Even if you do not use the Unicode encoding scheme for your data, you should be aware that Db2 uses Unicode in many of its internal processes. This use might affect your applications, queries, storage, and performance.

Db2 uses Unicode in the following ways:

Application preparation and processing:

- DBRMs that are produced in supported Db2 releases are stored in Unicode UTF-8.
- Db2 parses DBRMs in Unicode UTF-8, regardless of the original Db2 release that produced the DBRM.
- Db2 converts application source code to Unicode UTF-8 before it is processed by the precompiler. The precompiler then parses the source code in UTF-8. SQL statements and literals are considered part of the application source and are also parsed in UTF-8. SQL statement text is converted to UTF-8 if it is not already in UTF-8.

Db2 objects and data:

- Most Db2 catalog data is encoded in UTF-8. (The data in string columns that are not FOR BIT DATA columns in Unicode tables in the catalog is in UTF-8.) When you query the catalog, be aware that many string columns are VARCHAR(128). This data type and length enable you to easily port applications that run on other operating systems.
- The names of plans and packages are stored in Unicode UTF-8.
- The values of some special registers are stored in Unicode UTF-8.
- All EXPLAIN table data is encoded in Unicode UTF-8.
- SYSIBM.SYSDUMMYU is encoded in Unicode UTF-8. For more information about the SYSIBM.SYSDUMMY*x* tables, see SYSDUMMYx tables (Introduction to Db2 for z/OS).

Authorization:

- Db2 authorization processes work on Unicode data. When using certain external authorization processes, such as RACF, Db2 needs to convert the data to EBCDIC.

Traces:

- You can specify that Db2 return trace data in Unicode.

SQL statement processing:

- If you join Unicode and non-Unicode tables, Db2 performs some operations in Unicode. For example, if you compare columns from a Unicode table and an EBCDIC table, Db2 performs the comparison in Unicode.

Utility control statements:

- Utilities can process control statements that are written in Unicode UTF-8.

DRDA:

- Remote client systems can send and receive DRDA command and reply messages with character type data in Unicode (UTF-8).

Related concepts

DRDA character type parameters in Unicode

Remote Db2 applications can send and receive DRDA command and reply message parameters that contain character type data encoded in Unicode CCSID 1208 (UTF-8). Using Unicode instead of EBCDIC for these DRDA parameters can improve performance and avoid potential character conversion errors.

Related tasks

Specifying a CCSID for your application

© Copyright IBM Corp. 2003, 2020

In Db2 for z/OS applications, one CCSID is associated with the source code and one or more CCSIDs can be associated with the data that your application manipulates. The CCSID that Db2 associates with the data is called the *application encoding scheme*.

Setting up Db2 to ensure that it interprets characters correctly
You need to make sure that Db2 uses the correct code page (which is identified by a CCSID) to interpret your data. Otherwise, Db2 might store or use incorrect data. This situation is most likely to occur when characters are converted or transferred between systems.

Related reference
Descriptions of SQL processing options (Db2 Application programming and SQL)

Retrieving data from the Db2 catalog

Beginning in DB2 Version 8, most of the Db2 catalog data is stored in UTF-8. However, when you query the catalog data, Db2 converts the data to the application encoding scheme.

About this task

GUPI

Having the catalog in Unicode enables your SQL statements to use names and literals that contain characters that are not included in the subsystem EBCDIC CCSID.

Although most of the catalog is stored in UTF-8, several catalog tables are exceptions. SYSIBM.SYSCOPY in DSNDB06.SYSTSCPY is not stored in UTF-8. Also the non-Unicode SYSIBM.SYSDUMMY*x* tables are not stored in UTF-8.

You can select data from the catalog regardless of the application encoding scheme.

Procedure

To retrieve data from the Db2 catalog, perform all of the following actions:

- Ensure that you anticipate the sequence of the query result. Because the Db2 catalog is in Unicode UTF-8 , queries against the catalog return the data according to the Unicode sorting sequence. In Unicode, numeric characters are sorted before alphabetic characters.
- If you are using application host variables to store the results of any catalog string column values, ensure that these variables are large enough to hold those values.

  Many string columns in the Db2 catalog are VARCHAR(128). GUPI

Related tasks
Specifying a CCSID for your application
In Db2 for z/OS applications, one CCSID is associated with the source code and one or more CCSIDs can be associated with the data that your application manipulates. The CCSID that Db2 associates with the data is called the *application encoding scheme*.

Related reference
Differences between Unicode and EBCDIC sorting sequences
In Unicode, numeric characters are sorted before alphabetic characters. In EBCDIC, alphabetic characters are sorted before numeric characters.

Specifying that IFCID output should be in Unicode

You can start a performance, accounting, statistics, auditing, or monitoring trace by specifying the appropriate instrumentation facility component identifier (IFCID) in the START TRACE command. Many of these IFCIDs can write UTF-8 fields in the trace output.

Procedure

Set the UIFCIDS subsystem parameter to YES.

This parameter is called UIFCIDS in DSN6SYSP. It is also Option 11 (Unicode IFCIDS) on installation panel DSNTIPN. The default value is NO.

Results

Only a subset of the IFCID character fields are encoded in Unicode UTF-8. Those fields are identified in the IFCID record definition by a %U in the comment area to the right of the field declaration in the DSNDQWxx copy files.

If the UIFCIDS subsystem parameter is set to NO, the fields that are identified with %U are displayed in EBCDIC.

Related reference

-START TRACE (Db2) (Db2 Commands)

DSNTIPN: Tracing parameters panel (Db2 Installation and Migration)

Chapter 3. Setting up Db2 to ensure that it interprets characters correctly

You need to make sure that Db2 uses the correct code page (which is identified by a CCSID) to interpret your data. Otherwise, Db2 might store or use incorrect data. This situation is most likely to occur when characters are converted or transferred between systems.

Procedure

To ensure that Db2 interprets characters correctly:

1. Determine the CCSID of your data sources.
2. Based on the CCSIDs of your data sources, specify the correct CCSIDs for your subsystem, objects, and applications in Db2. If the CCSIDs of all of your data sources do not match and you need help determining the appropriate CCSIDs to specify, call IBM Support.

   Recommendation: If possible, set up your system, applications, and objects to avoid character conversion on z/OS, because character conversion has an expensive CPU cost. You can avoid character conversion by using the same CCSID in all of your data sources. Of course, do not do so at the expense of data integrity.
3. Set up z/OS Unicode Services.
4. Optional: Define any additional character conversions.

   Character conversions are already defined in the following two places:

   The Db2 catalog table SYSIBM.SYSSTRINGS
   This table contains character conversion definitions from IBM. You might have also added your own.

   The conversion image in z/OS Unicode Services
   You configured this image when you set up z/OS Unicode Services.

   However, you might need to define additional conversions. If you are not sure if a particular character conversion is defined to Db2, check your character conversion definitions.

What to do next

Recommendation: If your Db2 subsystem has users that use different CCSIDs, be careful when you create and name objects. Choose identifiers, such as table names and column names, that can be represented on all clients that access the Db2 subsystem.

How Db2 performs character conversions

When character conversions are needed, Db2 for z/OS performs these conversions automatically based on the CCSIDs of the source and target data. When you set up Db2, you need to identify valid conversion definitions for source and target CCSIDs to Db2. Some of these definitions are predefined for you.

To perform a conversion from one CCSID to another CCSID, Db2 uses the translation tables that are identified by the following resources in the order listed.

1. The Db2 catalog table SYSIBM.SYSSTRINGS

   Each row in this catalog table describes a conversion from one coded character set to another. IBM supplies some of the rows. You can also add your own rows. If the same pair of CCSIDs are in two rows, one row that is IBM-supplied and one row that you added, Db2 uses the row that you provided. Rows that you add have IBMREQD=N. However, some rows that have IBMREQD=N might have been loaded from maintenance that IBM ships between releases.

   SYSIBM.SYSSTRINGS describes only those conversions to and from ASCII and EBCDIC CCSIDs. Conversions to and from Unicode CCSIDs are not included in SYSIBM.SYSSTRINGS.

© Copyright IBM Corp. 2003, 2020

2. z/OS Unicode Services

   z/OS Unicode Services uses the conversion definitions in a conversion image data set.

Thus, any conversions that are defined in SYSIBM.SYSSTRINGS override the conversions that are defined in z/OS Unicode Services. If SYSIBM.SYSSTRINGS does not define a conversion, Db2 uses z/OS Unicode Services.

If a conversion for a certain combination of source and target CCSIDs is not defined in SYSIBM.SYSSTRINGS or z/OS Unicode Services, z/OS Unicode Services dynamically adds the conversion. This ability is available in z/OS 1.7 and later.

Related tasks

Setting up z/OS Unicode Services for Db2 for z/OS
Db2 for z/OS uses z/OS Unicode Services to perform character conversions and case conversions. You should set up z/OS Unicode Services specifically for Db2 for z/OS to ensure optimal Db2 performance.

Defining additional character conversions
You must define valid conversion definitions for source and target CCSIDs for Db2 to use when performing character conversions. Many of these definitions already exist in SYSIBM.SYSSTRINGS and in the conversion image that you set up for z/OS Unicode Services. However, you might need to add more.

Related reference

Manually setting up Unicode Services (z/OS: Unicode Services User's Guide and Reference)
Creating user-defined conversion tables (z/OS: Unicode Services User's Guide and Reference)
Conversion Tables Supplied with z/OS Unicode (z/OS: Unicode Services User's Guide and Reference)

SYSIBM.SYSSTRINGS catalog table

The Db2 catalog table SYSIBM.SYSSTRINGS contains information about valid character conversion definitions. Each row of SYSSTRINGS contains information about the conversion of character strings from one CCSID to another CCSID. Db2 uses the conversion tables that are identified by these rows.
GUPI

Db2 automatically performs any required conversions from the CCSID that is identified by the INCCSID column to the CCSID that is identified by the OUTCCSID column.

Restriction: You cannot update or delete rows that are provided by IBM. These rows are identified by a value of Y in the IBMREQD column. However, you can add another row with the same pair of CCSIDs. Rows that you add are identified by a value of N in the IBMREQD column. If two rows exist for the same pair of CCSIDs (an IBM-supplied row and a row that you added) Db2 uses your row for the conversion.

In some cases, rows with IBMREQD = N are not rows that you added. Sometimes, these rows have been supplied by IBM Software Support. For an example of adding such rows, look at job DSNTEJ1T.

Example: Assume that the SYSSTRINGS table includes the following rows:

```
     INCCSID     OUTCCSID  TRANSTYPE  ERRORBYTE  SUBBYTE  TRANSPROC      IBMREQD  TRANSTAB
---------+---------+---------+---------+---------+---------+---------+---------+-------------
        500           37  SS         ---------  -------                 Y        ...........
         37          500  SS         ---------  -------                 Y        ...........
        948           37  PS         3E         3F                      Y        ...........
```

All of these rows were supplied by IBM because they have the value Y in the IBMREQD column. These rows have the following meanings:

- The first row describes the conversion from CCSID 500 to CCSID 37.
- The second row describes the conversion from CCSID 37 to CCSID 500.
- The third row describes a conversion from CCSID 948 to CCSID 37 in which X'3E' is used as an error indicator and X'3F' is used as a substitute code point.

Tip: Use the HEX function to display the values of the ERRORBYTE, SUBBYTE, and TRANSTAB columns.
GUPI

Related concepts

Installation verification phases and programs (Db2 Installation and Migration)

Related reference

SYSSTRINGS catalog table ()

Finding the CCSID values of your data sources

Before you can specify appropriate CCSID values to Db2, you must know the CCSID values that are in effect for all of your data sources. Determining these CCSID values is the first step to preserving data integrity.

About this task

You should know the CCSIDs of all the data that Db2 handles, including all input and output sources, such as the following sources:

- local input and output devices, such as your 3270 terminal emulators and printers
- tape data
- source and data from your application, which are handled by either the Db2 precompiler or a compiler and the Db2 coprocessor
- data from gateway products, such as WebSphere® MQ, IMS Connect, CICS Transaction Gateway and any third party products
- FTP data
- Any data from a distributed environment

Procedure

Use the resources in the following table:

Table 13. How to find the CCSID of your data source

Source	Where find the CCSID in effect
Application data (in host variables or parameter markers)	Look at the value of the ENCODING bind option unless that value was overridden. For more details about how this option could have been overridden, see “Specifying a CCSID for your application” on page 54.

C/C++ application source code	Look at the Db2 precompiler or compiler listing for the CCSID options that were used. For example, the following listing for the Db2 precompiler shows that the application uses CCSID 1047:
	`OPTIONS USED - SPECIFIED OR DEFAULTED`
	`APOST`
	`APOSTSQL`
	`ATTACH(TSO)`
	`CCSID(1047)`

CICS Transaction Gateway	Look at the value of the system initialization parameter CLINTCP. See the following resources: • CLINTCP (CICS Transaction Server for z/OS) • Character data (CICS Transaction Server for z/OS) • ECI applications (CICS Transaction Gateway V5 The WebSphere Connector for CICS)

Table 13. How to find the CCSID of your data source (continued)

Source	Where find the CCSID in effect

COBOL application source code	Look at the Db2 precompiler or compiler listing for the CCSID options that were used. For example, the following listing for the Db2 precompiler shows that the application uses CCSID 37:
	`OPTIONS USED - SPECIFIED OR DEFAULTED`
	`APOST`
	`APOSTSQL`
	`ATTACH(TSO)`
	`CCSID(37)`

FTP	See How can I check my CCSIDs for FTP?.
IMS	Look at the terminal emulator CCSID. (Follow the instructions for ISPF or Personal Communications.) IMS uses this CCSID when communicating to Db2 for z/OS. In IMS Connect, conversion is done by user message exits. Look at those exits for CCSID information. See User exit (EX) ADD command (IMS Connect Extensions).
ISPF	Look at the value of the ISPF session variable ZTERMCID under ISPF option 7.3 - variable settings.
Personal Communications	Look at the **Host Code-Page** session parameter to find the terminal CCSID. See Configuring Sessions (Personal Communications)

PL/I application source code	Look at the Db2 precompiler or compiler listing for the CCSID options that were used. For example, the following listing for the Db2 precompiler shows that the application uses CCSID 37:
	`OPTIONS USED - SPECIFIED OR DEFAULTED`
	`APOST`
	`APOSTSQL`
	`ATTACH(TSO)`
	`CCSID(37)`

QMF	Check your Graphical Data Display Manager (GDDM) code page setting, because QMF uses GDDM to do the display. You can check your GDDM code page setting by looking at the APPCPG parameter, which can be set in one of the following two places: • a defaults module that is called ADMADFT • a file that is referred to as ADMDEFS. If no value is specified for APPCPG, GDDM uses a default CCSID of 00351. For more information about APPCPG, see External defaults: full descriptions (GDDM System Customization and Administration Guide). **Recommendation:** For QMF, set the APPCPG parameter to match the CCSID that is used by Db2 and your terminal emulator.
Queue Managers in WebSphere MQ	Follow the instructions for viewing and setting the Queue Manager CCSID in Data Conversion under WebSphere MQ. You can also check individual MQGET and MQPUT statements. These statements can override the MQ CCSID setting by specifying a CCSID in the statement.

Table 13. How to find the CCSID of your data source (continued)

Source	Where find the CCSID in effect
TSO	Perform one of the following actions: • Specify the CODEPG keyword when issuing the GTTERM -- Get Terminal Attributes (TSO/E Programming Services) macro to retrieve the Character Set and Code Page (CGCSGID) for a TSO session. • Issue the DISPLAY TSOUSER command (z/OS Communications Server: SNA Operation). The output from this command includes the CDCSGID information when it is available.
z/OS DFSMS SMS (the file system)	CCSID is an attribute of SMS-managed data sets. For more information about how that CCSID is set, see Data Conversion for z/OS Distributed FileManager (z/OS Distributed FileManager Guide and Reference) or Converting data for z/OS Network File System (z/OS Network File System Guide and Reference) or search the CCSID file tagging information in z/OS UNIX System Services Command Reference. However, the access methods (VSAM, BSAM/QSAM, BPAM, etc) for these data sets do not have support to perform conversions. Only DFM supports conversions between CCSIDs for DASD data sets. The CCSID value also can be used when reading or writing magnetic tapes that have ISO/ANSI tape labels. You can code the CCSID keyword on the DD statement or supply it in the data class. You can also supply the CCSID value on the JOB or STEP JCL statement. For more information, see Character Data Conversion (z/OS DFSMS Using Data Sets).

Related concepts

Differences between the Db2 precompiler and the Db2 coprocessor (Db2 Application programming and SQL)

Related reference

Internationalization: Locales and Character Sets (XL C/C++ Programming Guide)

Compiler Options (C/C++) (XL C/C++ User's Guide)

Planning to modify compiler option default values (COBOL) (Enterprise COBOL for z/OS Customization Guide)

Compiler options (COBOL) (Enterprise COBOL for z/OS Programming Guide)

Z variables (ISPF session variables)

Changing the default options (PL/I) (Enterprise PL/I for z/OS Programming Guide:)

Compile-time option descriptions (PL/I) (Enterprise PL/I for z/OS Programming Guide:)

WebSphere MQ library

Specifying CCSIDs in Db2

You must communicate to Db2 the correct CCSIDs to use for your data to ensure that Db2 correctly interprets your data. You can specify default subsystem CCSIDs. You can also specify CCSIDs for individual applications and Db2 objects.

About this task

Specifying appropriate CCSIDs also ensures that Db2 performs accurate character conversions when distributed systems access Db2.

Procedure

To specify CCSIDs in Db2:

- When you install Db2, specify default subsystem CCSIDs.
- When you create objects, specify object CCSIDs.
- When you create applications, specify application CCSIDs.

Related concepts
Euro symbol support (Db2 Installation and Migration)

Specifying subsystem CCSIDs

You specify the default subsystem CCSIDs when you install Db2. Db2 uses these values for objects and applications if no other CCSID values are specified.

Before you begin
Before you specify subsystem CCSIDs, determine the CCSID of your data sources. Knowing the CCSID of your data sources helps you determine what the subsystem CCSIDs should be. Ideally, your subsystem CCSIDs should match the CCSIDs in the majority of your data sources. If you need help determining the correct values, contact IBM Support.

About this task

Important: Never change CCSIDs on an existing Db2 subsystem without guidance from IBM Support. If you think you need to change your subsystem CCSIDs, first consider the effects on all of your tools, applications, and utilities. Then contact IBM Support.

Procedure

When you install Db2, specify the CCSIDs for the subsystem by using installation panel DSNTIPF or installation job DSNTIJUZ.

- For the subsystem EBCDIC and ASCII CCSIDs, you must specify values according to the following criteria:
  - You must specify valid, non-zero CCSIDs for single-byte character set (SBCS) data. You should specify the CCSID values that you want to use for EBCDIC and ASCII data and objects by default. For a list of valid CCSIDs, see EBCDIC and ASCII support.
  - If you use languages with double-byte characters, such as Chinese, Japanese, or Korean, you must also specify valid, non-zero CCSIDs for multibyte character set (MBCS) data and double-byte character set (DBCS) data. All three of these values, the single-byte CCSID value (SBCS), the mixed CCSID value, and the double-byte CCSID value (DBCS), that are associated with a particular encoding scheme are collectively called a *CCSID set*. If you set these three values by using the installation panel DSNTIPF, you need to explicitly specify only the MBCS value. Db2 calculates the value of the other two based on the MBCS value. If you specify these values in job DSNTIJUZ, you need specify all three values.
- For the subsystem Unicode CCSIDs, the values are provided for you, and you cannot change them. These CCSIDs are the only ones that Db2 uses for Unicode objects.

All of these CCSIDs are stored in *dsnhdecp* and must be valid. *dsnhdecp* is the DSNHDECP module or a user-specified application defaults module.

During startup processing, if Db2 detects invalid CCSID values, Db2 issues a message and terminates.

Related concepts
Job DSNTIJUZ and the subsystem parameter load module, application defaults load module, and DSNHMCID (Db2 Installation and Migration)
Euro symbol support (Db2 Installation and Migration)
Related reference
DSNTIPF: Application programming defaults panel 1 (Db2 Installation and Migration)
EBCDIC and ASCII support (Db2 Installation and Migration)

Subsystem CCSIDs and encoding schemes

Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

The subsystem CCSIDs are listed in the following table.

Table 14. Subsystem CCSIDs

Subsystem CCSID	Field on installation panel DSNTIPF where the value is set	Description	Corresponding *dsnhdecp* values[1, 3]
Subsystem default ASCII CCSID	ASCII CCSID	Specifies the default CCSID for ASCII-encoded character data that is stored in your Db2 subsystem or data sharing group. For a MIXED=NO subsystem, specify the ASCII SBCS CCSID only. In this case, the mixed and graphic CCSIDs are set to 65534 in *dsnhdecp*. For a MIXED=YES subsystem, specify the ASCII mixed CCSID. Based on the value that you entered, Db2 determines the SBCS CCSID and graphic CCSID. [2]	• ASCCSID (for single-byte data) • AMCCSID (for mixed data) • AGCCSID (for graphic data)
Subsystem default EBCDIC CCSID	EBCDIC CCSID	Specifies the default CCSID for EBCDIC-encoded character data that is stored in your Db2 subsystem or data sharing system. For a MIXED=NO subsystem, specify the EBCDIC SBCS CCSID only. In this case, the mixed and graphic CCSIDs are set to 65534 in *dsnhdecp*. For a MIXED=YES subsystem, specify the EBCDIC mixed CCSID. Based on the value that you entered, Db2 determines the SBCS CCSID and graphic CCSID. [2]	• SCCSID (for single-byte data) • MCCSID (for mixed data) • GCCSID (for graphic data)

Table 14. Subsystem CCSIDs (continued)

Subsystem CCSID	Field on installation panel DSNTIPF where the value is set	Description	Corresponding *dsnhdecp* values[1, 3]
Subsystem default Unicode CCSID	UNICODE CCSID	Specifies the default CCSID for Unicode character data that is stored in your Db2 subsystem or data sharing system. This field is pre-filled with the default value of 1208, which is the CCSID for UTF-8. You cannot change this value.	Because the value of UNICODE CCSID is always 1208, the *dsnhdecp* values are always as follows: • USCCSID (for single-byte data): 367 • UMCCSID (for mixed data): 1208 • UGCCSID (for graphic data): 1200

notes:

1. The three CCSID values, one for SBCS, one for mixed, and one for graphic, are called a *CCSID set*.
2. Whether the subsystem is a MIXED=YES subsystem or MIXED=NO subsystem depends on the value that you specified for the MIXED field on the same panel when you installed Db2. MIXED=NO is the default setting.

   Recommendation: Do not change the MIXED value after you install Db2.
3. *dsnhdecp* is the DSNHDECP module or a user-specified application defaults module.

The subsystem encoding schemes are listed in the following table.

Table 15. Subsystem encoding schemes

Subsystem encoding scheme	Field on installation panel DSNTIPF where the value is set	Description	Corresponding *dsnhdecp* values[1]
Subsystem default encoding scheme	DEF ENCODING SCHEME	Specifies which default subsystem CCSID (ASCII, EBCDIC, or Unicode) Db2 is to use for objects.	ENSCHEME
Subsystem default application encoding scheme	APPLICATION ENCODING	Specifies which default subsystem CCSID (ASCII, EBCDIC, or Unicode) Db2 is to use for application data.	APPENSCH

notes:

1. *dsnhdecp* is the DSNHDECP module or a user-specified application defaults module.

Related tasks

Determining current subsystem CCSID and encoding scheme values
For an existing subsystem, you can check the CCSID values, but do not make changes. If you suspect that the specified CCSIDs are incorrect or you need to change them, contact IBM Support.

Related reference

DSNTIPF: Application programming defaults panel 1 (Db2 Installation and Migration)

Determining current subsystem CCSID and encoding scheme values

For an existing subsystem, you can check the CCSID values, but do not make changes. If you suspect that the specified CCSIDs are incorrect or you need to change them, contact IBM Support.

Procedure

To determine current subsystem CCSID and encoding scheme values, perform one of the following actions:

- Use the GETVARIABLE function.

  GUPI

  Example GETVARIABLE calls: In all of the following examples, :hv3 is a varying-length character variable with a maximum length of 20.

  – The following example code retrieves the value of the subsystem EBCDIC CCSID:

  ```
  SET :hv3 = GETVARIABLE('SYSIBM.SYSTEM_EBCDIC_CCSID');
  ```

  The GETVARIABLE function returns three comma-delimited values that correspond to the SBCS, MIXED, and GRAPHIC CCSIDs for the encoding scheme.

  – The following example code retrieves the value of the subsystem default encoding scheme:

  ```
  SET :hv3 = GETVARIABLE('SYSIBM.ENCODING_SCHEME');
  ```

  – The following example code retrieves the value of the subsystem default application encoding scheme:

  ```
  SET :hv3 = GETVARIABLE('SYSIBM.APPLICATION_ENCODING_SCHEME');
  ```

  GUPI

- Run the DSNJU004 utility for the current subsystem or member, and look at the SYSTEM CCSIDS section in the output.

  Restriction: DSNJU004 does not return the subsystem encoding scheme values (DECP values ENSCHEME and APPENSCH). To get those values, use the GETVARIABLE function.

  Example: The following code shows example JCL to execute DSNJU004 and the relevant portion of the output.

  ```
  //PLM EXEC PGM=DSNJU004
  //GROUP DD DSN=DBD1.BSDS01,DISP=SHR
  //SYSPRINT DD SYSOUT=*
  //SYSIN DD *
  MEMBER *
  /*
  …
  SYSTEM CCSIDS
  18:12:47 MAY 18, 2005
  SYSTEM CCSIDS
  --------------------
  ASCII SBCS = 1252
  ASCII MIXED = 65534
  ASCII DBCS = 65534
  EBCDIC SBCS = 37
  EBCDIC MBCS = 65534
  EBCDIC DBCS = 65534
  UNICODE SBCS = 367
  UNICODE MBCS = 1208
  UNICODE DBCS = 1200
  DSNJ200I DSNJU004 PRINT LOG UTILITY PROCESSING COMPLETED SUCCESSFULLY
  ```

  This output shows that the default ASCII CCSID is 1252 and the default EBCDIC CCSID is 37. This subsystem does not have CCSIDs defined for ASCII or EBCDIC data that is mixed or double-byte. The Unicode CCSIDs are the default CCSIDs that are predefined by Db2. You cannot change these values.

- Run job DSNTEJ6Z, which calls the ADMIN_INFO_SYSPARM stored procedure to list your current subsystem parameter settings.

To determine the subsystem CCSIDs, examine the values of the following subsystem parameters:

- MIXED
- AGCCSID
- AMCCSID
- ASCCSID
- GCCSID
- MCCSID
- SCCSID
- UGCCSID
- UMCCSID
- USCCSID

Related concepts

Job DSNTEJ6Z (Db2 Installation and Migration)

Related reference

Subsystem CCSIDs and encoding schemes

Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

GETVARIABLE (Db2 SQL)

DSNJU004 (print log map) (Db2 Utilities)

Specifying object CCSIDs

The default encoding scheme for all Db2 objects is the value of ENSCHEME in *dsnhdecp*. *dsnhdecp* is the DSNHDECP module or a user-specified application defaults module. However, you can override this value for a particular object.

About this task

The ENSCHEME value was set during installation on panel DSNTIPF in the DEF ENCODING SCHEME field. Do not change the ENSCHEME DECP value without first considering the implications. This value controls the default encoding scheme of any newly created objects.

Procedure

Use the CCSID clause in the CREATE statement for any of the following objects:

- Database
- Table space
- Table
- Procedure or function

You can specify one of the following values in the CCSID clause:

ASCII
Use the subsystem default ASCII CCSID.

EBCDIC
Use the subsystem default EBCDIC CCSID.

Unicode
Use the subsystem default Unicode CCSID.

If you do not specify the CCSID clause, the object uses the subsystem default encoding scheme value (ENSCHEME in *dsnhdecp*).

Related tasks

Creating a Unicode table
If you plan to store Unicode data, create Unicode tables. If you try to insert Unicode data into an ASCII or EBCDIC table, data might be lost, unless you use escaped data.

Related reference

Subsystem CCSIDs and encoding schemes
Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

DSNTIPF: Application programming defaults panel 1 (Db2 Installation and Migration)

Setting up z/OS Unicode Services for Db2 for z/OS

Db2 for z/OS uses z/OS Unicode Services to perform character conversions and case conversions. You should set up z/OS Unicode Services specifically for Db2 for z/OS to ensure optimal Db2 performance.

About this task

z/OS Unicode Services uses a conversion image to determine how to handle various conversions. The conversion image tells z/OS Unicode Services which conversion tables to load and use for character and case conversions. This task explains how to set up such a conversion image. Starting in z/OS 1.7, if at any time Db2 needs a conversion that is not in your image, z/OS Unicode Services loads it on demand.

Tip: Even though you are not required to create your own conversion image, do so anyway by performing the steps in this task. Db2 for z/OS requires that certain conversions be available before it can start. When you define your own conversion image, as described in this task, those conversions are loaded when z/OS is IPLed and are available when Db2 starts. Otherwise, Db2 might be suspended by z/OS multiple times during startup as each of the required conversion tables is loaded by z/OS on demand.

Procedure

Follow the instructions in these sections in the z/OS Unicode Services information:

- Manually setting up Unicode Services (z/OS: Unicode Services User's Guide and Reference)
- Creating user-defined conversion tables (z/OS: Unicode Services User's Guide and Reference)

What to do next

If you later need to alter your conversion image in any way, use the SETUNI command.

Related reference

Conversion Tables Supplied with z/OS Unicode (z/OS: Unicode Services User's Guide and Reference)

Conversion image

A *conversion image* is a data set that contains the information that z/OS Unicode Services needs when performing character and case conversions. The conversion image defines which conversion tables z/OS is to load and use for these conversions.

You create a conversion image by invoking the z/OS Unicode Services image generator when you set up z/OS Unicode Services. The image generator creates the conversion image according to what you specify in the SYSIN DD statement in the job that invokes the image generator, CUNJIUTL. The generated image is stored in the data set that is identified in the SYSIMG DD statement. Messages from this process are listed in the data set that is identified by the SYSPRINT DD statement.

You can activate a conversion image during IPL or by issuing the z/OS SET UNI or SETUNI system command.

If you have z/OS 1.7 or later, conversions are loaded on request. However, to avoid waiting while conversion tables are loaded, you can create your own conversion image.

You can create more than one conversion image. These images are kept in different data sets. Use the SET UNI or SETUNI command to merge these images into the existing z/OS Unicode Services conversion image. Any tables in the new image that intersect with tables in the existing image are not loaded.

You can add, delete, or replace conversion images by using the SET UNI or SETUNI command.

Related reference

z/OS SETUNI Command (z/OS MVS System Commands)

Creating a conversion image (z/OS: Unicode Services User's Guide and Reference)

Basic character conversions for Db2 in the z/OS conversion image

When you set up z/OS Unicode Services for Db2, you need to define a set of basic conversions between various CCSIDs.

To define these basic character conversions, add the basic conversion statements to *high-level-qualifier*.SCUNJCL(CUNJIUTL). These conversions are then added to the z/OS conversion image. Any duplicate statements are ignored.

In these CONVERSION statements, the variables have the following meanings:

your sccsid
The EBCDIC SBCS CCSID that is specified in your *dsnhdecp* module.

your asccsid
The ASCII SBCS CCSID that is specified in your *dsnhdecp* module.

your mccsid
The EBCDIC MBCS CCSID that is specified in your *dsnhdecp* module.

your amccsid
The ASCII MBCS CCSID that is specified in your *dsnhdecp* module.

your gccsid
The EBCDIC graphic CCSID that is specified in your *dsnhdecp* module.

your agccsid
The ASCII graphic CCSID that is specified in your *dsnhdecp* module.

client ccsid
The CCSID from a client that makes remote connections to this Db2 subsystem.

dsnhdecp is the user-specified application defaults module.

Basic CONVERSION statements

- Specify the following conversion definitions between your ASCII and EBCDIC system CCSIDs and CCSIDs 367, 1208, and 1200:

```
CONVERSION your sccsid,00367,ER;
CONVERSION your sccsid,01200,ER;
CONVERSION your sccsid,01208,ER;
CONVERSION your sccsid,your asccsid,ER;
CONVERSION 00367,your sccsid,ER;
CONVERSION 00367,01200,ER;
CONVERSION 00367,01208,ER;
CONVERSION 00367,your asccsid,ER;
CONVERSION 01200,your sccsid,ER;
CONVERSION 01200,00367,ER;
CONVERSION 01200,01208,ER;
CONVERSION 01200,your asccsid,ER;
CONVERSION 01208,your sccsid,ER;
CONVERSION 01208,00367,ER;
CONVERSION 01208,01200,ER;
CONVERSION 01208,your asccsid,ER;
CONVERSION your asccsid,your sccsid,ER;
CONVERSION your asccsid,00367,ER;
CONVERSION your asccsid,01200,ER;
CONVERSION your asccsid,01208,ER;
```

- If you use the samples that are provided with Db2, also define the following conversions:

```
CONVERSION 00037, 00367, ER;
CONVERSION 00037, 01200, ER;
CONVERSION 00037, 1208, ER;
CONVERSION 00367, 0037, ER;
CONVERSION 01200, 00037, ER;
CONVERSION 1208, 00037, ER;
CONVERSION 01047, 00367, ER;
CONVERSION 01047, 01200, ER;
CONVERSION 01047, 1208, ER;
CONVERSION 00367, 1047, ER;
CONVERSION 01200, 1047, ER;
CONVERSION 1208, 1047, ER;
```

- **Optional:** For completeness, define the following conversions between CCSID 37 and CCSID 1047:

```
CONVERSION 00037, 01047, ER;
CONVERSION 001047, 0037, ER;
```

- If your *dsnhdecp* module specifies an EBCDIC SBCS CCSID other than CCSID 37 or CCSID 1047, define the following conversions:

```
CONVERSION your sccsid, 00367, ER;
CONVERSION your sccsid, 01200, ER;
CONVERSION your sccsid, 01208, ER;
CONVERSION 00367, your sccsid, ER;
CONVERSION 01200, your sccsid, ER;
CONVERSION 01208, your sccsid, ER;
```

- **Optional:** For completeness, define the following conversions between the EBCDIC SBCS CCSID that is defined in your *dsnhdecp* module and CCSIDs 37 and 1047.

```
CONVERSION 00037, your sccsid, ER;
CONVERSION your sccsid, 00037, ER;
CONVERSION 01047, your sccsid, ER;
CONVERSION your sccsid, 01047, ER;
```

- If your Db2 subsystem uses mixed-byte or double-byte CCSIDs for EBCDIC and ASCII, specify the following conversions:

```
CONVERSION your sccsid,your mccsid,ER;
CONVERSION your sccsid,your amccsid,ER;
CONVERSION your mccsid,00367,ER;
CONVERSION your mccsid,01200,ER;
CONVERSION your mccsid,01208,ER;
CONVERSION your mccsid,your sccsid,ER;
CONVERSION your mccsid,your asccsid,ER;
CONVERSION your mccsid,your amccsid,ER;
CONVERSION your gccsid,00367,ER;
CONVERSION your gccsid,01200,ER;
CONVERSION your gccsid,01208,ER;
CONVERSION your gccsid,your agccsid,ER;
CONVERSION your asccsid,your mccsid,ER;
CONVERSION your asccsid,your amccsid,ER;
CONVERSION your amccsid,your mccsid,ER;
CONVERSION your amccsid,00367,ER;
CONVERSION your amccsid,01200,ER;
CONVERSION your amccsid,01208,ER;
CONVERSION your amccsid,your asccsid,ER;
CONVERSION your amccsid,your sccsid,ER;
CONVERSION your agccsid,your gccsid,ER;
CONVERSION your agccsid,00367,ER;
CONVERSION your agccsid,01200,ER;
CONVERSION your agccsid,01208,ER;
CONVERSION 00367,your mccsid,ER;
CONVERSION 00367,your gccsid,ER;
CONVERSION 00367,your amccsid,ER;
CONVERSION 00367,your agccsid),ER;
CONVERSION 01200,your mccsid,ER;
CONVERSION 01200,your gccsid),ER;
CONVERSION 01200,your amccsid,ER;
CONVERSION 01200,your agccsid),ER;
CONVERSION 01208,your mccsid,ER;
CONVERSION 01208,your gccsid,ER;
```

```
CONVERSION 01208,your amccsid,ER;
CONVERSION 01208,your agccsid,ER;
```

- If your *dsnhdecp* module specifies an EBCDIC SBCS CCSID other than CCSID 37, specify the following conversions:

```
CONVERSION 00037,00367,ER;
CONVERSION 00037,00500,ER;
CONVERSION 00037,01047,ER;
CONVERSION 00037,01200,ER;
CONVERSION 00037,01208,ER;
CONVERSION 00037,(your asccsid),ER;
CONVERSION 00367,00037,ER;
CONVERSION 01200,00037,ER;
CONVERSION 01208,00037,ER;
CONVERSION your asccsid,00037,ER;
```

- If your *dsnhdecp* module specifies an EBCDIC SBCS CCSID other than CCSID 500, specify the following conversions:

```
CONVERSION 00500,00037,ER;
CONVERSION 00500,00367,ER;
CONVERSION 00500,01047,ER;
CONVERSION 00500,01200,ER;
CONVERSION 00500,01208,ER;
CONVERSION 00500,your asccsid,ER;
CONVERSION 00367,00500,ER;
CONVERSION 01200,00500,ER;
CONVERSION 01208,00500,ER;
CONVERSION your asccsid,00500,ER;
```

- If your *dsnhdecp* module specifies an EBCDIC SBCS CCSID other than CCSID 1047, specify the following conversions:

```
CONVERSION 01047,00037,ER;
CONVERSION 01047,00367,ER;
CONVERSION 01047,00500,ER;
CONVERSION 01047,01200,ER;
CONVERSION 01047,01208,ER;
CONVERSION 01047,your asccsid,ER;
CONVERSION 00367,01047,ER;
CONVERSION 01200,01047,ER;
CONVERSION 01208,01047,ER;
CONVERSION your asccsid,01047,ER;
```

- Define the following conversions for each additional CCSID that is presented by clients that make remote connections to this Db2 subsystem:

```
CONVERSION client ccsid,00367,ER;
CONVERSION client ccsid,01200,ER;
CONVERSION client ccsid,01208,ER;
CONVERSION 00367,client ccsid,ER;
CONVERSION 01200,client ccsid,ER;
CONVERSION 01208,client ccsid,ER;
```

Character conversions for Chinese, Japanese, and Korean character sets in the z/OS conversion image

If you use Chinese, Japanese, or Korean character sets, you need to specify several conversions for z/OS Unicode Services in addition to the basic conversions.

To define these conversions add the additional conversion statements to *high-level-qualifier*.SCUNJCL(CUNJIUTL). These conversions are then added to the z/OS conversion image. Any duplicate statements are ignored.

In these CONVERSION statements, the variables have the following meanings:

your sccsid
The EBCDIC SBCS CCSID that is specified in your *dsnhdecp* module.

your mccsid
The EBCDIC MBCS CCSID that is specified in your *dsnhdecp* module.

your gccsid
The EBCDIC DBCS CCSID that is specified in your *dsnhdecp* module.

your asccsid
The ASCII SBCS CCSID that is specified in your *dsnhdecp* module.

your amccsid
The ASCII MBCS CCSID that is specified in your *dsnhdecp* module.

your agccsid
The ASCII DBCS CCSID that is specified in your *dsnhdecp* module.

dsnhdecp is the user-specified application defaults module.

Additional CONVERSION statements

- Specify the following conversions between your EBCDIC MBCS CCSID and the Unicode CCSIDs:

```
CONVERSION your mccsid, 00367, ER;
CONVERSION your mccsid, 01200, ER;
CONVERSION your mccsid, 01208, ER;
CONVERSION 00367, your mccsid, ER;
CONVERSION 01200, your mccsid, ER;
CONVERSION 01208, your mccsid, ER;
```

- Specify the following conversions between your EBCDIC DBCS CCSID and the Unicode CCSIDs:

```
CONVERSION your gccsid, 00367, ER;
CONVERSION your gccsid, 01200, ER;
CONVERSION your gccsid, 01208, ER;
CONVERSION 00367, your gccsid, ER;
CONVERSION 01200, your gccsid, ER;
CONVERSION 01208, your gccsid, ER;
```

- Specify the following conversions between your ASCII SBCS CCSID and the Unicode CCSIDs:

```
CONVERSION your asccsid, 00367, ER;
CONVERSION your asccsid, 01200, ER;
CONVERSION your asccsid, 01208, ER;
CONVERSION 00367, your asccsid, ER;
CONVERSION 01200, your asccsid, ER;
CONVERSION 01208, your asccsid, ER;
```

- **Optional:** For completeness, specify the following conversions between your ASCII SBCS CCSID and CCSID 37, and between your ASCII SBCS CCSID and CCSID 1047:

```
CONVERSION 00037, your asccsid, ER;
CONVERSION your asccsid, 00037, ER;
CONVERSION 01047, your asccsid, ER;
CONVERSION your asccsid, 01047, ER;
```

- Specify the following conversions between your ASCII MBCS CCSID and the Unicode CCSIDs:

```
CONVERSION your amccsid, 00367, ER;
CONVERSION your amccsid, 01200, ER;
CONVERSION your amccsid, 01208, ER;
CONVERSION 00367, your amccsid, ER;
CONVERSION 01200, your amccsid, ER;
CONVERSION 01208, your amccsid, ER;
```

- Specify the following conversions between your ASCII DBCS CCSID and the Unicode CCSIDs:

```
CONVERSION your agccsid, 00367, ER;
CONVERSION your agccsid, 01200, ER;
CONVERSION your agccsid, 01208, ER;
CONVERSION 00367, your agccsid, ER;
CONVERSION 01200, your agccsid, ER;
CONVERSION 01208, your agccsid, ER;
```

- If your *dsnhdecp* module specifies an EBCDIC SBCS CCSID other than CCSID 37 or CCSID 1047, specify the following conversions:

```
CONVERSION your sccsid, your asccsid, ER;
CONVERSION your asccsid, your sccsid, ER;
```

- **Optional:** Specify the following conversions between your system EBCDIC MBCS CCSID and ASCII MBCS CCSID and between your system EBCDIC DBCS CCSID and your ASCII DBCS CCSID:

```
CONVERSION your mccsid, your amccsid, ER;
CONVERSION your amccsid, your mccsid, ER;
CONVERSION your gccsid, your agccsid, ER;
CONVERSION your agccsid, your gccsid, ER;
```

Defining additional character conversions

You must define valid conversion definitions for source and target CCSIDs for Db2 to use when performing character conversions. Many of these definitions already exist in SYSIBM.SYSSTRINGS and in the conversion image that you set up for z/OS Unicode Services. However, you might need to add more.

Procedure

Add the definition to one of the following places:

- The Db2 catalog table SYSIBM.SYSSTRINGS

  Insert a row with the appropriate definition. The definitions in this table take precedence over the definitions in z/OS Unicode Services with several exceptions, which are described after this list. Rows that you insert have a value of 'N' in the IBMREQD column and take precedence over the IBM-supplied rows.

- z/OS Unicode Services

  You can either load a new conversion image that contains the conversion definitions or add a single conversion definition to the existing image. For instructions on how to load or alter conversion images, see “Setting up z/OS Unicode Services for Db2 for z/OS” on page 35.

Related concepts

How Db2 performs character conversions
When character conversions are needed, Db2 for z/OS performs these conversions automatically based on the CCSIDs of the source and target data. When you set up Db2, you need to identify valid conversion definitions for source and target CCSIDs to Db2. Some of these definitions are predefined for you.

SYSIBM.SYSSTRINGS catalog table
The Db2 catalog table SYSIBM.SYSSTRINGS contains information about valid character conversion definitions. Each row of SYSSTRINGS contains information about the conversion of character strings from one CCSID to another CCSID. Db2 uses the conversion tables that are identified by these rows.

Checking defined character conversions

Character conversion definitions identify valid conversions for source and target CCSIDs for Db2. Many of these definitions are predefined. If you are not sure if a particular character conversion that you need is defined to Db2, check your character conversion definitions.

Procedure

To check defined character conversions:

1. Query the Db2 catalog table SYSIBM.SYSSTRINGS.

   Each row in the catalog table describes a conversion from one CCSID to another. IBM supplies some of the rows. You can also add your own rows.

   GUPI

Example: You can use the following query to view the defined conversions for CCSID 500:

```
SELECT INCCSID, OUTCCSID, TRANSTYPE, HEX(ERRORBYTE) AS ERRORBYTE,
       HEX(SUBBYTE) AS SUBBYTE, TRANSPROC, IBMREQD, HEX(TRANSTAB) AS TRANSTAB
       FROM SYSIBM.SYSSTRINGS WHERE CCSID=500
```

GUPI

2. Check the conversion image for z/OS Unicode Services by using the DISPLAY UNI command.

   This image contains character conversion definitions. If a definition for a particular source CCSID and target CCSID already exists in SYSIBM.SYSSTRINGS, Db2 uses that definition instead. The exception is for Unicode CCSIDs. If the source or target CCSID is 1200 or 1208, Db2 uses the definition in the conversion image for z/OS Unicode Services

   For an example of the DISPLAY UNI output, see "Unicode CCSIDs" on page 13

Related concepts

How Db2 performs character conversions

When character conversions are needed, Db2 for z/OS performs these conversions automatically based on the CCSIDs of the source and target data. When you set up Db2, you need to identify valid conversion definitions for source and target CCSIDs to Db2. Some of these definitions are predefined for you.

SYSIBM.SYSSTRINGS catalog table

The Db2 catalog table SYSIBM.SYSSTRINGS contains information about valid character conversion definitions. Each row of SYSSTRINGS contains information about the conversion of character strings from one CCSID to another CCSID. Db2 uses the conversion tables that are identified by these rows.

Related reference

z/OS Displaying Unicode Services (z/OS MVS System Commands)

Chapter 4. Storing Unicode data

Db2 for z/OS supports the full Unicode *character repertoire*, or set of characters. You can store Db2 data as UTF-8 or UTF-16.

Related concepts

Unicode

Unicode is an encoding scheme that currently provides a unique code point for over 100,000 characters. This standard enables systems to more easily handle global data, regardless of the platform, program, or language.

Related reference

UTFs

Deciding whether to store data as UTF-8 or UTF-16

If you create a Unicode database in Db2 for z/OS, you need to decide whether to use UTF-8 or UTF-16. Db2 for z/OS does not support storing data as UTF-32. UTF-8 and UTF-16 can both represent any Unicode character that you need to represent, but each format has advantages and disadvantages depending on your situation.

Procedure

Consider the following recommendations and guidelines:

- **Performance recommendation:** Store your data in Db2 in the same format as your application. This setup ensures optimal performance, because character conversion is avoided.

  This recommendation is especially important when the application is written in a language that runs on z/OS (for example COBOL on z/OS), because the CPU cost of character conversion on z/OS can be very expensive.

  Examples:

  - COBOL and PL/I on z/OS use UTF-16 or UTF-8 for Unicode data. Even if your compiler supports UTF-8 data, the optimal situation is to store your data in Db2 in UTF-16. UTF-16 data can potentially take more storage than UTF-8 data, but because no conversion occurs when you use UTF-16 data, you avoid a significant performance impact.
  - For Java applications that use IBM Data Server Driver for JDBC and SQLJ type 4 connectivity, which sends the data in UTF-8, store your data in Db2 as UTF-8 data.

  If you have both local and remote applications on different operating systems, choose the format based on the encoding of the local application.

- **Storage recommendation:** After you consider performance, consider your storage requirements. Store the data in the format that requires the least space for your data.

  UTF-16 does not always require more storage than UTF-8. The amount of storage that is required depends on your data. For example, Latin-1 characters always take 1 byte in UTF-8 and 2 bytes in UTF-16. However, Japanese characters take 3 to 4 bytes in UTF-8 and 2 to 4 bytes in UTF-16.

  Example: Db2 for z/OS uses UTF-8 for the catalog. Because the catalog contains mostly Latin-1 characters, this format uses considerably less space than UTF-16.

- **Recommendation for MQ, CICS Transaction Gateway, and IMS Connect messages:** When messages are passed from one technology to another, everything in the message is usually converted to characters. You should consider the size of these messages when you decide when and where to use certain UTFs. For example, suppose that you have COBOL applications, which use UTF-16, but you are concerned about the size of the messages. You might decide to convert the messages to UTF-8 before you put them on the wire. This setup compresses the messages.

© Copyright IBM Corp. 2003, 2020

What to do next

If you choose a Unicode format for performance reasons and are concerned about the extra storage that the format requires, see "Tips for handling any extra storage that Unicode data might require" on page 46.

Related reference

UTFs

Creating a Unicode table

If you plan to store Unicode data, create Unicode tables. If you try to insert Unicode data into an ASCII or EBCDIC table, data might be lost, unless you use escaped data.

About this task

Recommendation: When you create objects, use standard characters for the object names and column names. Unique characters, such as ü or é, can complicate your applications if conversions are needed.

Procedure

To create a Unicode table:

1. In the CREATE DATABASE, CREATE TABLESPACE, or CREATE TABLE statement, specify the CCSID UNICODE clause.

   By default, the encoding scheme of a table is the same as the encoding scheme of its table space. Also by default, the encoding scheme of the table space is the same as the encoding scheme of its database. You can override the encoding scheme with the CCSID clause in the CREATE TABLESPACE or CREATE TABLE statement. However, all tables within a table space must have the same CCSID.
2. In the CREATE TABLE statement, for each column definition, specify the appropriate data type, subtype, and length value.

   data type

   Use one of the following data types:

   - For UTF-8 data, create columns of type CHAR, VARCHAR, or CLOB.
   - For UTF-16 data, create columns of type GRAPHIC, VARGRAPHIC, or DBCLOB.
   - For binary data, create columns of type BINARY, VARBINARY, and BLOB.

   Recommendation: In general, use varying-length columns for Unicode tables because the number of bytes in a Unicode column usually is two to three times that of an EBCDIC column.

   The general guideline is to use variable length for columns that are greater than 18 bytes unless you know that the entire column is to always be filled. For example, if you store a timestamp in character form (not as the Db2 TIMESTAMP datatype), you need a column with some number of characters. In DB2 9, that number would be 26 characters. (In ASCII, EBCDIC, or UTF-8, that column is 26 bytes. In UTF-16, that column is 52 bytes.) Because the timestamp is always the same size, using a varying-length column does not save storage. However, suppose that you have a name field that is in ASCII or EBCDIC and allows for names of 26 characters. (In ASCII SBCS or EBCDIC SBCS, you use 26 bytes. In UTF-8, you need 78 bytes. In UTF-16, you need 52 bytes.) In this case, you want to use a varying-length column, because the name field is likely to have many blanks and you do not want to store them.

   subtype

   For character columns, optionally specify one the following subtypes for the column by adding the FOR *subtype* DATA clause to the column definition:

   SBCS

   Specify this subtype if the column is to contain only those UTF-8 characters that are stored as 1 byte. Those characters are the first 128 characters in the Unicode code page. Data that is stored in a SBCS character column in a Unicode table has a CCSID of 367.

MIXED
Specify this subtype if the column is to contain any UTF-8 data that is more than 1 byte. MIXED is the default value. Character data in a Unicode table is stored as mixed data by default, even if your subsystem is defined with a MIXED DECP value of NO. Data that is stored in a MIXED character column in a Unicode table has a CCSID of 1208.

BIT
This subtype specifies that the column contains BIT data. CCSID 66534 is associated with FOR BIT DATA columns.

Recommendation: Although you can also specify the subtype BIT for CHAR and VARCHAR columns that contain BIT data, use the BINARY or VARBINARY data types instead.

Do not use FOR BIT DATA columns for the sole purpose of handling international data. Only use FOR BIT DATA columns if you have a specific reason, such as encryption. Otherwise, this data type can cause problems. For example, if you have a string of length 10 and put it in a FOR BIT DATA column of length 12, Db2 pads the string with two blanks. The hexadecimal value that is used for those blanks is system specific. For example, X'40' is used for EBCDIC and X'20' is used for Unicode. These different hexadecimal values can potentially cause problems when you convert this data.

length
To determine the appropriate the length value, follow the instructions in "Estimating the column size for Unicode data" on page 47.

Db2 associates a certain CCSID with the column depending on the data type that you specify. The following table summarizes the possible column data types in a Unicode table and the CCSIDs that are associated with the data in those columns.

Table 16. CCSIDs that are associated with columns in a Unicode table

Column data type	Associated CCSID	Format in which the data is stored
CHAR"3.a" on page 45	1208	UTF-8
CHAR FOR SBCS DATA	367	7-bit ASCII
CHAR FOR MIXED DATA	1208	UTF-8
CHAR FOR BIT DATA	66534	NA
VARCHAR"3.a" on page 45	1208	UTF-8
VARCHAR FOR SBCS DATA	367	7-bit ASCII
VARCHAR FOR MIXED DATA	1208	UTF-8
VARCHAR FOR BIT DATA	66534	NA
CLOB"3.a" on page 45	1208	UTF-8
CLOB FOR SBCS DATA	367	7-bit ASCII
CLOB FOR MIXED DATA	1208	UTF-8
GRAPHIC	1200	UTF-16
VARGRAPHIC	1200	UTF-16
DBCLOB	1200	UTF-16

Note:

a. If you do not specify a subtype, Db2 assumes FOR MIXED DATA.

Example

GUPI The following CREATE TABLE statement creates a Unicode table.

```
CREATE TABLE UNITAB
 (C1 CHAR(4)FOR SBCS DATA,
  C2 CHAR(4),
  C3 GRAPHIC(4),
  C4 VARCHAR(4) FOR SBCS DATA,
  C5 VARCHAR(4),
  C6 VARGRAPHIC(4))
CCSID Unicode
```

GUPI

Columns C1 and C4 can contain only 1-byte UTF-8 data. (This data has CCSID 367 and is stored in 7-bit ASCII format.) Columns C2 and C5 can contain any UTF-8 data. Columns C3® and C6 can contain UTF-16 data.

The CHAR and VARCHAR columns each have a length of 4 bytes. That length means that each of these columns can contain one of the following characters or sets of characters:

- one UTF-8 character that is 4 bytes
- two UTF-8 characters that are each 2 bytes
- one 3-byte UTF-8 characters and one one-byte UTF-8 character
- four one-byte UTF-8 characters

The GRAPHIC and VARGRAPHIC columns each have a length of 4 UTF-16 code units. (A UTF-16 code unit is 16 bits or 2 bytes.) For UTF-16 characters that are 2 bytes, this length means 4 characters. However, this length does not always correlate to 4 characters. Consider supplementary UTF-16 characters, which are each 2 UTF-16 code units or 4 bytes. If you include any supplementary characters in the column, the column cannot include 4 characters. Thus, the length of this column can contain 2, 3, or 4 characters, depending on the size of the character. For example, each of these GRAPHIC and VARGRAPHIC columns can contain one of the following characters or sets of characters:

- four 2-byte UTF-16 characters
- two 4-byte UTF-16 characters
- one 4-byte UTF-16 character and two 2-byte UTF-16 characters

Related tasks

Generating escaped Unicode data
If you pass Unicode characters to an application or object that is not intended to handle Unicode data, data might be lost unless you escape certain characters. For example, you might need to pass Unicode data through an application that has EBCDIC host variables. Or you might want to store Unicode data in a non-Unicode table.

Related reference

UTFs
CREATE DATABASE (Db2 SQL)
CREATE TABLE (Db2 SQL)
CREATE TABLESPACE (Db2 SQL)
MIXED DATA field (MIXED DECP value) (Db2 Installation and Migration)

Tips for handling any extra storage that Unicode data might require

Unicode data often requires more storage than EBCDIC or ASCII data, but not always. The amount of extra storage that is required depends on the type of data and whether it is stored in UTF-8 or UTF-16 format.

Unicode data almost never requires double the amount of storage as EBCDIC or ASCII data. That amount of extra storage is the extreme worst-case scenario. To figure out how much space your Unicode data requires, consider the following two factors:

The type of data that you plan to store in Db2

How many character fields do you have? Any increased storage requirement affects mostly character fields. So if you convert an existing Db2 database to Unicode, look at the character fields that are defined in your existing database to get an idea of how much the database expands when you convert it to Unicode.

Is the data Latin-1, Japanese, Chinese, or something else? For example, the first 128 Latin-1 code points of UTF-8 take up only 1 byte. Those code points include the characters A-Z, a-z, and 0-9. Thus, these characters do not take up any more space in UTF-8 than they do in EBCDIC or ASCII. Also, consider that Chinese characters can take up less space in Unicode than EBCDIC.

The UTF format

Are you using UTF-8 or UTF-16? UTF-8 characters can take 1, 2, 3, or 4 bytes. UTF-16 characters can take 2 or 4 bytes. Even though UTF-16 often takes more storage, UTF-16 is sometimes a wiser choice for performance reasons. Also, in some cases, UTF-16 takes up less space. For example, Japanese characters are 3 or 4 bytes in UTF-8, but 2 or 4 bytes in UTF-16.

If possible, use the following general recommendations to minimize the storage impact of Unicode data:

- Use data compression.
- Use non-padded indexes. If you are converting data that has padded indexes to Unicode, change those indexes to be non-padded. This type of index can save index storage space.
- If a column length is more than 18 bytes, use variable length data types.
- Use 8-KB pages instead of the default 4-KB pages by increasing the size of the buffer pools. (The buffer pool in which you define the table space determines the page size.)

Related tasks

Compressing your data (Db2 Performance)

Saving disk space by using non-Padded indexes (Db2 Performance)

Related reference

UTFs

Estimating the column size for Unicode data

When you create a table to store Unicode data, allocate columns for storage length, not for display length.

Procedure

To estimate the column size for Unicode data, perform one of the following actions:

- For UTF-8 data, allocate three times the column size that you would allocate for a non-Unicode table.

  For example, if you use CHAR(10) for a name column in an EBCDIC table, use VARCHAR(30) for the same column in a Unicode table. This column can contain 30 bytes or ten 3-byte characters. In this case, use VARCHAR instead of CHAR, because the length (30) is greater than 18. (18 is traditionally the length when VARCHAR should be used instead of CHAR.)

  This estimate allows for the worst-case expansion of UTF-8 data. The worst case for SBCS data is that 1 byte in ASCII or EBCDIC expands to 3 bytes in UTF-8. For mixed data, such as Chinese, Japanese, or Korean characters, the same worst-case scenario applies. You might have 2-, 3- and 4-byte characters, depending on the encoding, that expand to a four-byte UTF-8 character in the worst case. However, because these characters used more than one byte in ASCII or EBCDIC, the worst-case expansion in UTF-8 is still three times the original size.
- For UTF-16 data, allocate two times the column size that you would allocate for a non-Unicode table, and use the GRAPHIC or VARGRAPHIC data types.

  For example, if you use CHAR(10) for a name column in an EBCDIC table, use VARGRAPHIC(10) for the same column in a Unicode table. CHAR(10) is 10 bytes long. VARGRAPHIC(10) is 20 bytes long or the equivalent of 10 two-byte characters.

Recommendation: If your application is written in COBOL or PL/I, store your data in UTF-16, and use the GRAPHIC and VARGRAPHIC data types. Thus, the Unicode format in your application matches the format in your database. This setup avoids conversion costs.

Related reference

UTFs

Inserting data into a Unicode table

Unicode tables can store any characters. For characters that you can type on your keyboard, INSERT statements are straightforward. But suppose that you want to insert a character that is not on your keyboard, such as the yen sign (¥) on the U.S. keyboard. That process requires some extra steps.

Procedure

To insert data into a Unicode table, use one of the following methods:

- Load the data from a data set by using the LOAD utility. If the input data set is already in Unicode, specify the UNICODE option. If the data is not in Unicode, ensure that you specify the appropriate encoding scheme keyword (ASCII, EBCDIC, or CCSID) in the LOAD utility statement. The default is EBCDIC. Db2 converts ASCII and EBCDIC data to Unicode when it is loaded into a Unicode table. Be aware that this conversion might cause the data to expand.
- Load the data from an another table by using the cross-loader function. If the data is from an EBCDIC or ASCII table, Db2 converts it to Unicode when it is loaded into the target Unicode table. Be aware that this conversion might cause the data to expand.
- Insert individual rows by using the INSERT statement. For characters that cannot be typed on your keyboard, use the Unicode constant UX'*xxxx*'.

  This constant is always in UTF-16, which means that you need to specify the value in UTF-16 format. To determine the Unicode constant for a particular character perform the following steps:

  a) Look up the Unicode code point. Use the Unicode character code charts on the Unicode Consortium web site. For example, the yen sign (¥) is U+00A5.

  b) Convert the Unicode code point to UTF-16 format by performing one of the following actions:

  – If the Unicode code point U+*yyyy* is less than U+FFFF, encoding it in UTF-16 is simple. Just copy the value. For example, the following Unicode code points can be specified as the following Unicode constants:

  Table 17. Unicode code points and their corresponding Unicode constants for Unicode code points that are less than U+FFFF

Character	Unicode code point	UTF-16 format	Unicode constant
¥	U+00A5	X'00A5'	UX'00A5'
K	U+0138	X'0138'	UX'0138'
✎	U+270E	X'270E'	UX'270E'

  – If the Unicode code point U+*yyyy* is greater than or equal to U+FFFF, encode that character as UTF-16 format, and use that encoded value. For example, Unicode code point U+200D0 can be encoded in UTF-16 as X'D840DCD0'. Thus, the Unicode constant is UX'D840DCD0'.

    You can find the steps for how to manually encode and decode Unicode data on the Unicode Consortium web site. Alternatively, you can use a converter tool to do the conversion for you.

Example: The following INSERT statement inserts a row with Unicode character U+200D0, which is 𠃐, in the second column.

```
INSERT INTO UNITAB VALUES ('7A907',UX'D840DCD0','A');
```

Related concepts

Expanding conversion
An *expanding conversion* occurs when the length of the converted string is greater than that of the source string.

Graphic string constants (Db2 SQL)

Related tasks

Loading data by using the cross-loader function (Db2 Utilities)

Related reference

LOAD (Db2 Utilities)

UTFs

Related information

UTF-8, UTF-16, UTF-32 & BOM (on Unicode Consortium website)

Inserting Unicode data into a non-Unicode table

If you insert Unicode data into an EBCDIC or ASCII table, use escaped data for those characters that cannot be represented in the target encoding scheme. Using escaped data ensures that those characters are preserved.

Procedure

To insert Unicode data into a non-Unicode table, perform one of the following actions:

- If the target table is ASCII, use the ASCII_STR function to generate escaped data for those characters that do not exist in ASCII.
- If the target table is EBCDIC, use the EBCDIC_STR function to generate escaped data for those characters that do not exist in EBCDIC.

Related concepts

Potential problems when inserting non-Unicode data into a Unicode table
If you insert EBCDIC or ASCII data into a Unicode table, the data is converted to Unicode. The length of this converted data might increase so much that it causes the operation to fail.

Related tasks

Generating escaped Unicode data
If you pass Unicode characters to an application or object that is not intended to handle Unicode data, data might be lost unless you escape certain characters. For example, you might need to pass Unicode data through an application that has EBCDIC host variables. Or you might want to store Unicode data in a non-Unicode table.

Related reference

ASCII_STR or ASCIISTR (Db2 SQL)

EBCDIC_STR (Db2 SQL)

Converting existing Db2 data to Unicode

If your database and applications handle international data, consider converting your Db2 data to Unicode. Using Unicode might prevent character conversions and thus improve performance and help ensure data integrity. However, Unicode data might require more space. Depending on the data, these characters can be two to three times the size of EBCDIC or ASCII characters.

Before you begin

Before you convert your existing Db2 data to Unicode, think about the following items:

- Consider the affects on all associated applications and tools. For example, consider the affects on any green screen applications.

- Understand where the data originally came from. If the data was already converted from its original form, it might contain substitution characters. If so, consider converting the data back to its original form and then converting it to Unicode.

  For example, suppose that you convert data from EBCDIC to Unicode, and the data was originally in ASCII. You might need to convert the data to its original ASCII format and then to Unicode. Do this extra conversion if the original ASCII data underwent a round-trip conversion to EBCDIC and not all of the characters exist in EBCDIC. For example, suppose that you converted data from ASCII CCSID 1252 to EBCDIC CCSID 37. CCSID 1252 contains characters that do not exist in CCSID 37. Thus, the EBCDIC data has control characters in place of any characters that existed in ASCII but not in EBCDIC. (Consider the example of the trademark symbol ™ in "Round-trip conversion" on page 18.) Converting the data to ASCII first recovers the original values before you convert to Unicode.

Procedure

To convert existing Db2 data to Unicode:

1. Create one or more Unicode tables for this data.
2. Use one of the following techniques to load the existing data into your new Unicode tables. Db2 converts the data to Unicode when it loads it.
   - Use the INSERT statement with a subselect.

     Example:

     ```
     INSERT INTO UNICODETABLE
          SELECT *
          FROM EBCDICTABLE;
     ```

     For this example, make sure that the columns for both UNICODETABLE and EBCDICTABLE are compatible. For example, if the first column of EBCDICTABLE is a character column, the first column of UNICODETABLE should also be a character column; if the second column of EBCDICTABLE is a numeric column, the second column of UNICODETABLE should also be a numeric column.
   - Use the UNLOAD utility to unload the data as is into an EBCDIC or ASCII data set. Then, LOAD the data into your new Unicode table. Specify the appropriate encoding scheme keyword (ASCII, EBCDIC, or CCSID) in the LOAD statement.

     Recommendation: Use the PUNCHDDN option of the UNLOAD utility to generate corresponding LOAD utility statements for the data as Db2 unloads it.

     Example: The following example JCL performs the following actions:

     – STEP1 creates and populates two tables. T1 is a Unicode table. T2 is an EBCDIC table.
     – STEP2 unloads the data from EBCDIC table T2. The UNLOAD statement contains the PUNCHDDN option. This option generates (in the SYSPUNCH data set) corresponding LOAD statements to load the data back into the original table, T2. To use this SYSPUNCH file to load the unloaded data to table T1, you must modify the SYSPUNCH or JCL.
     – STEP3 then loads the data that was unloaded in STEP2 into Unicode table T1. Because the catalog defines the table as Unicode, the data is converted to Unicode when it is loaded.
     – STEP4 outputs the current data in both tables.

     ```
     //STEP1    EXEC  TSOBATCH
     //SYSTSIN  DD *
     DSN S(SSTR) R(1) T(1)
     RUN PROGRAM DSNTEP2 PLAN(DSNTEPB1)
     END
     //SYSIN DD *
         DROP DATABASE DB1;
         COMMIT;
         CREATE DATABASE DB1 CCSID UNICODE;
         CREATE TABLESPACE TS1 IN DB1;
         CREATE TABLE T1 (C1 CHAR(7)) IN DB1.TS1;

         DROP DATABASE DB2;
         COMMIT;
     ```

```
    CREATE DATABASE DB2 CCSID EBCDIC;
    CREATE TABLESPACE TS2 IN DB2;
    CREATE TABLE T2 (C1 CHAR(7)) IN DB2.TS2;
    INSERT INTO T2 VALUES ('ABCDEFG');

    INSERT INTO T1 (SELECT * FROM T2);

    SELECT * FROM SYSADM.T1;
    SELECT * FROM SYSADM.T2;
/*
//************************************************
//STEP2    EXEC DSNUPROC,UID='SMPLUNLD',UTPROC='',SYSTEM='SSTR'
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)
//SYSUT2   DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)
//SORTOUT  DD UNIT=SYSDA,SPACE=(CYL,(5,5),RLSE)
//SYSIN    DD *
  TEMPLATE REC DSN TEST123.STEP2.UNLOAD.SYSREC SPACE(15,5)
           CYL UNIT(3390) VOLUMES(SCR03)
  TEMPLATE CARD DSN TEST123.STEP2.UNLOAD.SYSPUNCH SPACE(15,5)
           CYL UNIT(3390) VOLUMES(SCR03)
  UNLOAD DATA FROM TABLE SYSADM.T2
     (C1  CHAR(7))
         UNLDDN REC PUNCHDDN CARD SHRLEVEL CHANGE
/*
/*
//*********************************************************************
//* DSNUPROC UTILITY STEP
//*********************************************************************
//STEP3    EXEC DSNUPROC,UID='LI848.LOAD1',TIME=1440,
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSUT1   DD DSN=TEST123.STEP3.LOAD.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=TEST123.STEP3.LOAD.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSMAP   DD DSN=TEST123.STEP3.LOAD.SYSMAP,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN    DD *
TEMPLATE CWPH5APV
     DSN('TEST123.STEP2.UNLOAD.SYSREC')
     DISP(OLD,KEEP,KEEP)
LOAD DATA INDDN CWPH5APV LOG NO  RESUME YES
  EBCDIC CCSID(00037,00000,00000)
 INTO TABLE
 "SYSADM".
 "T1"
 WHEN(00001:00002) = X'0006'
 NUMRECS                    1
 ( "C1"
  POSITION(  00004:00010) CHAR MIXED(007)
                          NULLIF(00003)=X'FF'
 )
//*******************************************************************
//STEP4    EXEC TSOBATCH,DB2LEV=DB2A
//SYSTSIN  DD *
DSN SYSTEM(SSTR)
RUN PROGRAM DSNTEP2 PLAN(DSNTEPB1))
//SYSIN    DD *
SELECT * FROM SYSADM.T1;
SELECT * FROM SYSADM.T2;
/*
```

- Use the cross-loader function to load the output of a dynamic SQL SELECT statement into your new Unicode table. The SELECT statement selects the entire table.

  Example: In the following example, assume that table T1 is in Unicode and table T2 is in EBCDIC. This example uses a cursor to select all data from T2 and then load it into T1. This process is known as the *cross-loader function*. The data is converted to Unicode when it is loaded.

```
//STEP5    EXEC DSNUPROC,UID='LOADIT',TIME=1440,COND=(EVEN),
//         UTPROC='',
//         SYSTEM='SSTR',DB2LEV=DB2A
//SYSUT1   DD DSN=TEST123.STEP5.LOAD.SYSUT1,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SORTOUT  DD DSN=TEST123.STEP5.LOAD.SORTOUT,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
//SYSCOPY  DD DSN=TEST123.STEP5.LOAD.COPY,DISP=(MOD,DELETE,CATLG),
//         UNIT=SYSDA,SPACE=(4000,(20,20),,,ROUND)
```

```
//SYSIN    DD *
EXEC SQL
  DECLARE C1 CURSOR FOR SELECT C1 FROM SYSADM.T2
ENDEXEC

LOAD DATA REPLACE INCURSOR C1 INTO TABLE SYSADM.T1

//*******************************************************************
//STEP1    EXEC TSOBATCH,DB2LEV=DB2A
//SYSTSIN  DD *
DSN SYSTEM(SSTR)
RUN PROGRAM DSNTEP2 PLAN(DSNTEPB1)
//SYSIN    DD *
SELECT * FROM SYSADM.T1;
SELECT * FROM SYSADM.T2;
/*
```

3. Modify any SQL in your applications to account for length differences. If you use any length functions, such as CHARACTER_LENGTH and SUBSTRING, use the CODEUNITS16 and CODEUNITS32 options to specify how you want Db2 to calculate the length.

Related concepts

Round-trip conversion

A *round-trip conversion* ensures the integrity of all character data from the source CCSID to the target CCSID and back to the source. Even if the target CCSID does not support a given character, the character regains its original hexadecimal value after it is converted back to the source CCSID.

Related tasks

Generating escaped Unicode data

If you pass Unicode characters to an application or object that is not intended to handle Unicode data, data might be lost unless you escape certain characters. For example, you might need to pass Unicode data through an application that has EBCDIC host variables. Or you might want to store Unicode data in a non-Unicode table.

Specifying how Db2 calculates the length of a string

Loading data by using the cross-loader function (Db2 Utilities)

Related reference

INSERT (Db2 SQL)

LOAD (Db2 Utilities)

UNLOAD (Db2 Utilities)

Effects on access paths when converting data to Unicode

If you convert your data to Unicode, the access paths for queries on that data do not change simply because the data is now in Unicode. However, valid reasons might exist for a change in the access path for a Unicode table.

For example, the schemas are likely different in Unicode tables than EBCDIC tables. The index key might be longer. For example, an index might be 5 levels in Unicode instead of 4 levels in EBCDIC. Also, the number of rows per page might be fewer.

All of the regular rules for access paths and tuning queries still apply to Unicode tables.

Related tasks

Writing efficient SQL queries (Db2 Performance)

Chapter 5. Application programming with Unicode data and multiple CCSIDs

If your application handles Unicode data or data that is in different encoding schemes, you should be aware of several programming techniques and recommendations in Db2.

Db2 always returns data to your application in the CCSID that your application uses for data. This CCSID is called the *application encoding scheme*.

Recommendations: Use the following general recommendations to guide you in writing and preparing your application programs:

- If possible, use either Unicode or EBCDIC data, but not both. If you do choose to use multiple encoding schemes, consider the following possible implications for data loss and performance:
  - Managing multiple CCSIDs in your application can be difficult. To ensure that data is not lost, you have to control where the data goes, a path that potentially includes many modules.
  - Many environments, such as CICS Transaction Gateway and WebSphere MQ are message-based. In these cases, the entire message must be in a single encoding scheme. Because the entire message is in one encoding, flowing some data through the application in EBCDIC and some in Unicode makes little sense. You still have to convert all of it to a single encoding, such as Unicode, right before the putting the message on the wire.
  - Db2 tables must be in the same encoding scheme. You cannot make some columns Unicode and some EBCDIC. If your application processes some columns in Unicode and others in EBCDIC, character conversion occurs, which likely increases the performance overhead.
- If you are using Unicode data in COBOL or PL/I applications, use the coprocessor.
- If your COBOL, PL/I, C/C++ , or Assembler application handles Unicode data, do not place literals in the source code of the application. Because these language compilers do not support Unicode source code, they could misinterpret these literal values. Instead, place these literal values in a file or Db2 table that can be accessed at the start of the program to load the values. (Files and host variables are not precompiled and compiled as application source code.)
- If an expanding or contracting conversion occurs on your data, the length of the data might change. Be aware of these length changes when you use the LENGTH function, CHARACTER_LENGTH function, SUBSTRING function, and SUBSTR function on the converted string. For CHARACTER_LENGTH and SUBSTRING, use the CODEUNITS16 and CODEUNITS32 options to specify how you want Db2 to calculate the length.
- If you need to represent characters from multiple Latin-based character sets, such as Latin-1 and Latin-4, consider using Unicode for your application encoding scheme. An SBCS CCSID does not have enough code points to represent all of the characters that the combination of the two character sets require. For example, assume that your application uses an EBCDIC CCSID, such as 277 or 1069. You might have some data that is represented in the database in Unicode but that cannot be retrieved by the application without substitution. If your application needs to handle only one language at a time, you can set up your infrastructure in one of the following ways:
  - Have one version of your application that uses CCSID 277 and another version that uses CCSID 1069. Also have two corresponding subsystems, one that uses CCSID 277 and another that uses CCSID 1069. (You cannot have multiple EBCDIC CCSIDs in one Db2 subsystem.)
  - Store the data in Unicode and have one version of your application that uses CCSID 277 and another version that uses CCSID 1069. Then bind these applications with different values for the ENCODING bind option.
  - Store the data in Unicode and have one version of your application that uses an EBCDIC CCSID and another version that uses Unicode.

  However, if you require that a single version of the application handle both Latin-1 and Latin-4 character sets, your application needs to process data in Unicode.

© Copyright IBM Corp. 2003, 2020

Related concepts

Contracting conversion
A *contracting conversion* occurs when the length of the converted string is smaller than that of the source string.

Expanding conversion
An *expanding conversion* occurs when the length of the converted string is greater than that of the source string.

Application encoding schemes and Db2 ODBC (Db2 Programming for ODBC)

Related tasks

Specifying how Db2 calculates the length of a string

Generating table and view declarations by using DCLGEN (Db2 Application programming and SQL)

Related reference

LENGTH (Db2 SQL)

SUBSTR (Db2 SQL)

SUBSTRING (Db2 SQL)

Enterprise PL/I for z/OS

Related information

Guidelines to design global solutions

Application encoding scheme

The *application encoding scheme* is the CCSID that your application uses to interpret data in host variables. For Db2 for z/OS applications, typically the application encoding scheme is the value of the ENCODING bind option. (By default this value is the subsystem default application encoding scheme.)

However, you can also set the CCSID of application data by using the DECLARE VARIABLE statement with the CCSID option or the CURRENT APPLICATION ENCODING SCHEME special register. If you are using the Db2 coprocessor, you can use various language compiler options to override the Db2 application encoding scheme for an application. For detailed instructions on how to set the application encoding scheme, see “Specifying a CCSID for your application” on page 54.

Db2 automatically converts any data that you select to the application encoding scheme. For example, if you use SPUFI to select catalog data (which is in CCSID 1208), Db2 converts the data to the application CCSID of SPUFI. Your version of SPUFI should be bound with a CCSID that matches the CCSID of your terminal emulator. (By default, SPUFI is bound with CCSID 37. However, you can bind different versions of SPUFI with different CCSIDs.) Assume that you are following this good practice of having your SPUFI CCSID match your terminal emulator CCSID. In this case, any character in the selected data that does not exist in the CCSID of your terminal emulator is not displayed correctly. For example, if SPUFI and your terminal emulator are set to CCSID 37, the Euro symbol (€) can not be displayed.

Specifying a CCSID for your application

In Db2 for z/OS applications, one CCSID is associated with the source code and one or more CCSIDs can be associated with the data that your application manipulates. The CCSID that Db2 associates with the data is called the *application encoding scheme*.

About this task

If the CCSID values do not match the actual CCSID of the data or source, data corruption might occur.

Recommendation: Having all of your CCSIDs match is ideal but not always possible. If they do not match, character conversion can occur, and you should consider the possible consequences of character conversion.

Procedure

To specify a CCSID for your application:

1. Use the options as shown in the following table:

Table 18. Options to set application CCSIDs

Item for which you want to specify the CCSID	Option to use
Application source code (which includes SQL statements and literal strings in the SQL statements)	If you are using the Db2 precompiler, use the CCSID SQL processing option when you precompile the application. Specify the same CCSID when you compile the application. If you are using the Db2 coprocessor, use the language compiler to set the CCSID. For COBOL, PL/I, and C/C++, use the following instructions: "2.a" on page 56 • "Specifying CCSIDs for COBOL applications when using the Db2 coprocessor" on page 60 • "Specifying CCSIDs for PL/I applications when using the Db2 coprocessor" on page 63 • "Specifying CCSIDs for C/C++ applications when using the Db2 coprocessor" on page 66 The default CCSID for the application source code is the subsystem EBCDIC CCSID (DECP value SCCSID or MCCSID). Db2 uses this value if you do not use one of the preceding mechanisms to specify a CCSID. **Restriction:** The compilers for high level host languages do not support Unicode source code.
Application data (values that are passed through host variables and parameter markers) within SQL statements "2.b" on page 56	Use one or more of the following Db2 mechanisms to set the CCSID value of the application data, which is called the *application encoding scheme*: • Use the ENCODING bind option."2.c" on page 56 This option typically yields the best performance. • You can override the CCSID for a particular host variable by using the DECLARE VARIABLE statement with the CCSID option. • You can override the CCSID for parameter markers in dynamic SQL by specifying the CURRENT APPLICATION ENCODING SCHEME special register. Db2 uses the value of this special register at the time that the statement is executed. The default CCSID for the application data is the subsystem default application encoding scheme. For static SQL (host variables), this value is the APPENSCH value from the DECP that is loaded when you bind your application. For dynamic SQL (parameter markers), this value is the APPENSCH value from the DECP that is loaded at the time that the application is executed. Alternatively, if you are using the Db2 coprocessor on a COBOL or PL/I application, you can override the ENCODING bind option by using the following language compiler options: • "Specifying CCSIDs for COBOL applications when using the Db2 coprocessor" on page 60 • "Specifying CCSIDs for PL/I applications when using the Db2 coprocessor" on page 63
Application data that is referenced outside of SQL statements	Use the rules of the programming language. In some cases, the CCSID of this data is the same as the CCSID of the source code.

Table 18. Options to set application CCSIDs (continued)

Item for which you want to specify the CCSID	Option to use

Notes:

a. For older compilers that do not pass a CCSID value to the Db2 coprocessor, use the SQL compiler option with the CCSID suboption to specify a value.

b. You can specify different CCSIDs for different pieces of data in one application. However, if you specify multiple CCSIDs, do so with caution.

c. For DRDA applications, the ENCODING bind option does not set the CCSID of the data. In a DRDA environment, the CCSIDs are communicated as part of the protocol.

2. Optional: If you want to confirm which CCSID value the Db2 precompiler used, look at the precompiler listing. If you want to confirm which CCSID value the Db2 coprocessor used, look at the compiler listing.

   If you need help finding the CCSID values in these listings, see the example listings in "Finding the CCSID values of your data sources" on page 27.

   You can also use these listings to confirm which DECP module Db2 used. Knowing which DECP module is useful if you modified a DECP value, such as APPENSCH, before you compiled or executed your program. You can see which DECP module and which values Db2 used.

Related concepts

Objects with different CCSIDs in the same SQL statement
You can reference data with different CCSIDs from the same SQL statement. This ability is useful if you use table objects such as tables, views, temporary tables, query tables, and user-defined functions with different CCSIDs. However, you should understand how Db2 for z/OS processes these queries so that you can code them correctly.

Related reference

Descriptions of SQL processing options (Db2 Application programming and SQL)

Subsystem CCSIDs and encoding schemes
Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

Details of CCSID options for application programs

You have several options in Db2 to set CCSIDs for your applications.

For the overall context of when to use each option, see "Specifying a CCSID for your application" on page 54. The following list explains the details of each option.

CCSID SQL processing option

If you are using the Db2 precompiler, use this option to specify the CCSID in which the source program is written. This value ensures that the Db2 precompiler correctly parses the SQL statements and literal string values in those statements at precompile time. The default value is the subsystem EBCDIC CCSID (DECP value SCCSID if MIXED=NO or MCCSID if MIXED=YES).

Db2 converts the source code from the specified CCSID to Unicode UTF-8 before it is processed by the precompiler. The precompiler then parses the source code in Unicode UTF-8.

The value that you specify for the precompiler must match the value that you specify to the compiler when you compile the program.

If you are using the Db2 coprocessor, do not specify this option. Instead, use the language compiler options. The coprocessor uses the CCSID that is passed to it from the language compiler to convert the SQL statement text. If the compiler does not pass a CCSID, the Db2 coprocessor uses the CCSID suboption of the compiler SQL option. If that suboption is not specified, the Db2 coprocessor uses the subsystem EBCDIC CCSID (DECP value SCCSID or MCCSID) as the CCSID for the source.

ENCODING bind option

Use this option when you bind your application to specify the CCSID of the data in your application. This value applies to both host variables in static SQL statements and parameter markers in dynamic SQL statements unless this value is overridden. For example, you can override the CCSID of host variables by using certain language compiler options or by specifying a DECLARE VARIABLE statement with the CCSID option. You can override the CCSID of parameter markers in dynamic SQL statements by using the CURRENT APPLICATION ENCODING SCHEME special register.

This value can be EBCDIC, ASCII, Unicode, or a valid CCSID. If the value is EBCDIC, ASCII, or Unicode, Db2 uses the subsystem default CCSID for that encoding scheme.

The default value is the subsystem default application encoding scheme (the DECP value APPENSCH), which, by default, is EBCDIC. The Db2 sample applications are bound with ENCODING EBCDIC.

For example, some possible uses of the ENCODING bind option are as follows:

- You have a C/C++ program that accesses an ASCII library on z/OS. In this case, bind the program with ENCODING ASCII.
- You use QMF and have a data center in Germany and 3270 emulators in France. You might want to bind a special version of QMF for French by specifying ENCODING 1147.

In general, any time the CCSID of your source data does not match the subsystem default CCSID, use the ENCODING option to tell Db2 the correct CCSID. The source data can come from a terminal emulator, an MQ queue, or elsewhere.

If you use the Db2 coprocessor on a COBOL application that contains PIC X variables and specify the NOSQLCCSID compiler option, do not specify ENCODING UNICODE. If you specify this option, Db2 interprets these character variables as UTF-8, but COBOL does not support UTF-8.

In a DRDA environment, the CCSIDs are communicated as part of the protocol. Db2 does not use the ENCODING bind option to determine the CCSID of data from a remote application or to encode data to send to a remote application. However, the ENCODING bind option can influence internal Db2 processing. Db2 uses the value of this bind option when it processes SET statements or any statement that contains multiple CCSIDs. For example, Db2 uses the ENCODING option that was specified when the package was bound to evaluate the following statement:

```
SET :hv1 = SUBSTR(:hv_locator, 1, 100);
```

CURRENT APPLICATION ENCODING SCHEME special register

Use this special register to specify the CCSID for data that is passed through parameter markers in dynamic SQL statements. This value does not apply to static SQL statements.

You can set the value of this special register by using the SET CURRENT APPLICATION ENCODING SCHEME statement in your application program.

The value can be EBCDIC, ASCII, Unicode, or a valid CCSID. If the value is EBCDIC, ASCII, or Unicode, Db2 uses the subsystem default CCSID for that encoding scheme.

The default value is the value of the ENCODING bind option. For native SQL procedures, the default value is the APPLICATION ENCODING SCHEME option of the CREATE PROCEDURE or ALTER PROCEDURE statement. If you do not specify these values, the default value is the subsystem default application encoding scheme.

DECLARE VARIABLE statement with the CCSID option

Use this statement in your application to define a CCSID for a particular host variable. This value overrides the CURRENT APPLICATION ENCODING SCHEME special register value, the ENCODING bind option and any compiler and precompiler CCSID options.

The value for the CCSID option can be EBCDIC, ASCII, Unicode, or a valid CCSID. If the value is EBCDIC, ASCII, or Unicode, Db2 uses the subsystem default CCSID for that encoding scheme.

Use the DECLARE VARIABLE statement with the CCSID option when your application handles a piece of data that you know has a different CCSID.

In the case of bit data, no CCSID is needed. For this type of data, use a DECLARE VARIABLE statement so that no CCSID is associated with the variable, as shown in the following COBOL example:

```
EXEC SQL DECLARE : hv1 VARIABLE FOR BIT DATA END-EXEC.
```

If you are using DCLGEN, you can specify DCLBIT(YES) to create DECLARE VARIABLE statements for columns that are declared with the FOR BIT DATA clause. For example, the following DCLGEN output shows such a declaration for a COBOL application:

GUPI

```
******************************************************************
* DCLGEN TABLE(ADMF001.T1)                                       *
*        LIBRARY(USER.DBRMLIB.DATA(T1))                          *
*        LANGUAGE(COBOL)                                         *
*        QUOTE                                                   *
*        DBCSSYMBOL(N)                                           *
*        DCLBIT(YES)                                             *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS   *
******************************************************************
     EXEC SQL DECLARE ADMF001.T1 TABLE
     ( NAME                          VARGRAPHIC(15),
       ADDRESS                       VARGRAPHIC(25),
       …
       PASSWORD                      CHAR(8)
      ) END-EXEC.
******************************************************************
* DECLARED VARIABLES FOR 'FOR BIT DATA' COLUMNS                  *
******************************************************************
     EXEC SQL DECLARE
      :PASSWORD
     VARIABLE FOR BIT DATA END-EXEC.
******************************************************************
* COBOL DECLARATION FOR TABLE ADMF001.T1                         *
******************************************************************
 01  DCLT1.
     10 NAME.
        49 NAME-LEN          PIC S9(4) USAGE COMP.
        49 NAME-TEXT         PIC N(15).
     10 ADDRESS.
        49 ADDRESS-LEN       PIC S9(4) USAGE COMP.
        49 ADDRESS-TEXT      PIC N(25).
     10 CITY.
        49 CITY-LEN          PIC S9(4) USAGE COMP.
        49 CITY-TEXT         PIC N(20).
     10 STATE                PIC N(2).
     10 ZIP                  PIC N(5).
     10 PASSWORD             PIC (8).
******************************************************************
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 6       *
******************************************************************
```

GUPI

In this example, notice the DBCSSYMBOL option for DCLGEN. You can use this option to specify how you want COBOL graphic variables to be generated. If you plan to use Unicode variables and the Db2 coprocessor, you should specify DBCSSYMBOL(N) so that you get PIC N variables.

Related tasks

Generating table and view declarations by using DCLGEN (Db2 Application programming and SQL)

Related reference

Subsystem CCSIDs and encoding schemes

Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

Descriptions of SQL processing options (Db2 Application programming and SQL)

BIND and REBIND options for packages, plans, and services (Db2 Commands)

SET CURRENT APPLICATION ENCODING SCHEME (Db2 SQL)

DECLARE VARIABLE (Db2 SQL)

DCLGEN (DECLARATIONS GENERATOR) (DSN) (Db2 Commands)

Examples of specifying CCSIDs for application data

If your applications handle international or Unicode data, you probably need to specify a different application CCSID than the default. Also, if you deploy applications to international locations, you probably need to bind different versions of the application with the appropriate CCSIDs.

Example of ENCODING(UNICODE) bind option: Assume that the package MY_PACK is bound with the option ENCODING(UNICODE). Db2 assumes that all character input and output host variables are encoded using CCSID 1208. Db2 assumes that all graphic input and output host variables are encoded using CCSID 1200.

Example of setting CCSIDs in a distributed environment: Assume that your Db2 for z/OS subsystem is located in the United States and you have users around the world that connect to this subsystem. The following figure illustrates this scenario.

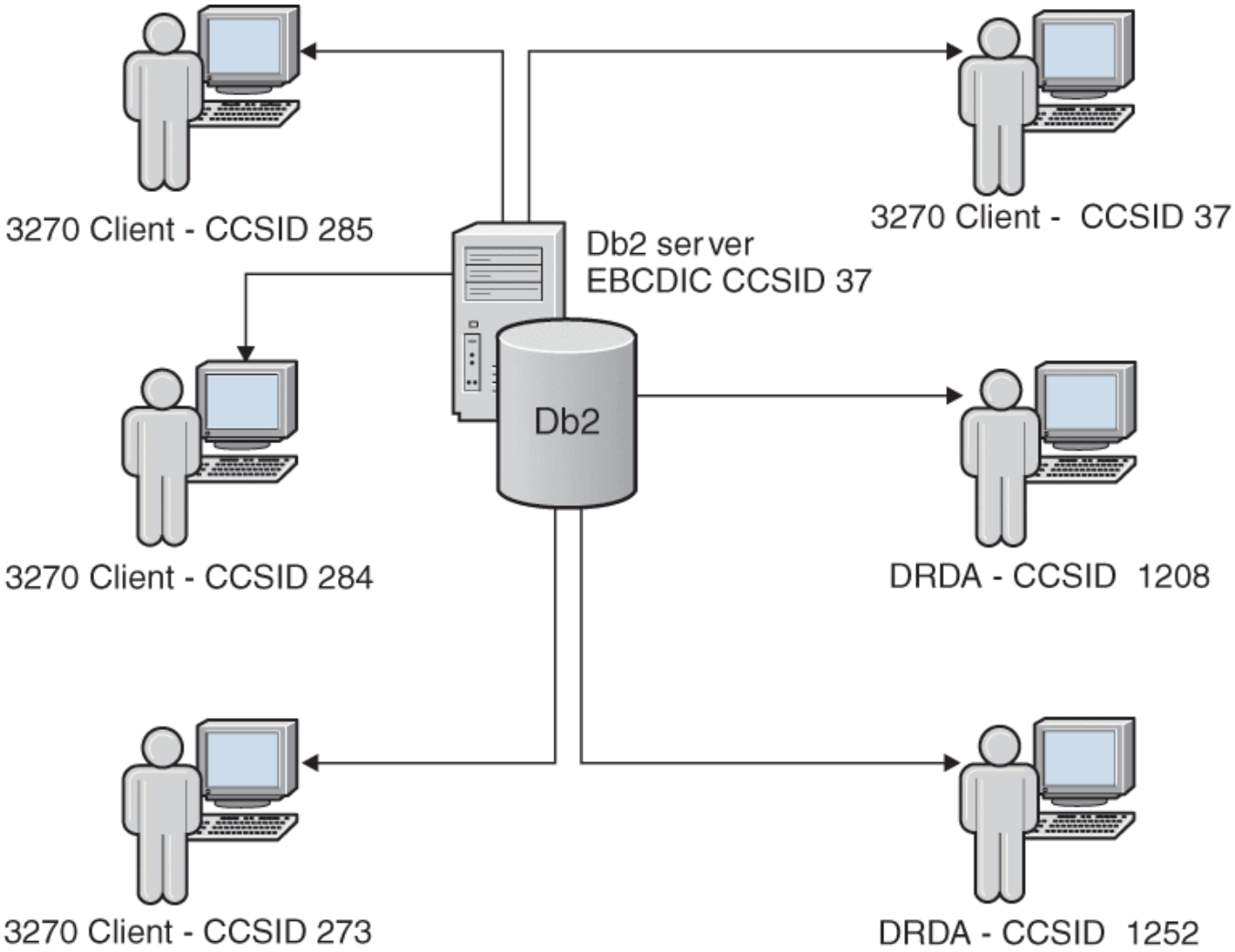


Figure 1. Example of setting CCSIDs in a distributed environment

The users that use DRDA do not need to use the ENCODING bind option to handle CCSID conversions because DRDA handles all conversions. However, users might choose to specify the ENCODING bind option to influence internal Db2 processing.

The users that use 3270 terminal emulators need to set up their emulators to use a CCSID that corresponds to the country in which they reside. In this example, the CCSID of one of those terminal emulators is 285. You need to bind the plans or packages that this client uses with ENCODING(285). Likewise, for the terminal emulator that has CCSID 284, you need to bind the plans or packages that this client uses with ENCODING(284). Also, for the terminal emulator that has CCSID 273, you need to bind the plans or packages that this client uses with ENCODING(273).

Example of ensuring that remote users access the correct version of the SPUFI application according to their terminal CCSID: Suppose that you want to prevent users in the U.K. from using the U.S. version of SPUFI. Instead of granting the EXECUTE privilege for the SPUFI packages to public, restrict access to only those users in the U.S. Then, bind additional SPUFI packages with the ENCODING bind option that specifies the appropriate terminal CCSID for the U.K. (For instructions on how to find the terminal CCSID value in ISPF, TSO, or CICS, see "Finding the CCSID values of your data sources" on page 27.) Authorize the U.K. users to use only this version of SPUFI.

Related tasks

Making SPUFI work with different terminal CCSIDs (Db2 Installation and Migration)

Related reference

Z variables (ISPF session variables)

Configuring Sessions (Personal Communications)

Specifying CCSIDs for COBOL applications when using the Db2 coprocessor

If you are using the Db2 coprocessor to prepare a COBOL application with SQL statements, use the COBOL compiler to specify the CCSID of the application source code. For optimal performance, use Db2 to specify the CCSID of the application data in SQL statements.

About this task

The COBOL compiler accepts only one CCSID value that it uses for both the application source code and data. However, Db2 can accept one CCSID value for the source code and one or more CCSID values for the data that is manipulated in SQL statements through host variables and parameter markers.

Procedure

To specify CCSIDs for COBOL applications when using the Db2 coprocessor:

1. To specify the CCSID of the COBOL application source code, use the CODEPAGE compiler option. [2]

   Example: Both of the following JCL EXEC statements for COBOL compile jobs specify a CCSID of 37:

   ```
   //COB  EXEC  PGM=IGYCRCTL,PARM='...,SQL,CODEPAGE(037),...
   ```

   ```
   //COB EXEC PGM=IGYCRCTL,PARM='...,SQL,CP(37),...
   ```

   Otherwise, if you do not specify the CODEPAGE compiler option, the default COBOL compiler CCSID is passed to the Db2 coprocessor and is used as the CCSID for the source code. The default COBOL compiler CCSID is 1140 unless you changed it.

   Example: The following JCL EXEC statement for a COBOL compile job does not explicitly specify a CCSID. In this case, the COBOL compiler passes the default CCSID, 1140, to the Db2 coprocessor.

   ```
   //COB EXEC PGM=IGYCRCTR,PARM='...,SQL(),...'
   ```

   CCSID 1140 is the equivalent to CCSID 37 plus the euro symbol (€). However, be aware that conversions, and thus conversion cost, still occur between CCSID 1140 and CCSID 37.

   Recommendation: If you are using the Db2 coprocessor on a COBOL application, do not specify the SQL compiler option with the CCSID suboption. If you specify it anyway, and it conflicts with the

[2] If you are using an older compiler that does not otherwise pass a CCSID value to Db2, use the SQL compiler option with the CCSID suboption to specify the CCSID of the application source. For example, the following EXEC statement for a COBOL compile job specifies a source CCSID of 1140.

```
//COB EXEC PGM=IGYCRCTR,PARM='SQL("CCSID(1140)"'
```

CODEPAGE compiler value, Db2 issues a warning. [3] For example, the following EXEC statement for a COBOL compile job specifies a CCSID value of 1140; the CCSID value 37 is ignored:

```
//COB EXEC PGM=IGYCRCTR,PARM='CP(1140),SQL("CCSID(37)")'
```

To verify which CCSID value was used, look at the compiler listing and check the CCSID option.

2. To specify whether Db2 or the COBOL compiler determines the CCSID of the data, specify one of the following compiler options:

   NOSQLCCSID (Recommended option)
   Specifies that the CCSID that is passed from the COBOL compiler is used only for the COBOL application source and string literals. That CCSID is not used for the host variables and parameter markers in SQL statements. For host variables and parameter markers, Db2 uses the CCSIDs that are specified through Db2 mechanisms, as described in the next step.

   Specifying NOSQLCCSID typically yields better performance, because you can then use a Db2 mechanism to specify the host variable CCSIDs.

   NOSQLCCSID simulates the behavior of the precompiler. Specify this option for existing applications that previously used the Db2 precompiler and now use the Db2 coprocessor. By default, the Db2 coprocessor uses the same CCSID value that is passed from the COBOL CODEPAGE(*nnnnn*) compiler option for both the source code and data. This behavior is different than the Db2 precompiler, which does not use the CCSID from the COBOL compiler. For applications that use the Db2 precompiler, you specify the CCSID for the data through Db2 mechanisms only. When you specify NOSQLCCSID, Db2 does not use the COBOL CCSID for the application data.

   SQLCCSID (default option)
   Specifies that the Db2 coprocessor is to use the CCSID from the COBOL CODEPAGE(*nnnnn*) compiler option for your application data. [3] The CCSID value is the same one that you specified for your COBOL source code.

3. Optional: If you specified the NOSQLCCSID compiler option, explicitly specify a value for the ENCODING bind option.

   The ENCODING bind option value is used as the CCSID for the application data. If you do not explicitly specify a value, the default value for the ENCODING bind option is used.

4. To override the CCSID for particular host variables or parameter markers, use the DECLARE VARIABLE statement or CURRENT APPLICATION ENCODING SCHEME special register. If you need help using either of these techniques, following the instructions in Specifying a CCSID for your application.

   If you specify different CCSIDs for different pieces of data, do so with caution.

5. If you want to specify a Unicode CCSID for a particular variable, declare it as a PIC N USAGE NATIONAL variable.

   For COBOL PIC N USAGE NATIONAL variables, the Db2 coprocessor always uses the CCSID 1200. You do not need to use a DECLARE VARIABLE statement with the CCSID clause for these variables.

Example

The following table shows examples of the CCSID that Db2 uses for data in COBOL applications depending on the options that you specify. This table assumes that you did not specify any DECLARE VARIABLE statements with the CCSID clause.

[3] The exception is if you are using an older compiler that does not otherwise pass a CCSID value to the Db2 coprocessor. In this case, you need to specify the SQL compiler option with the CCSID suboption.

Table 19. CCSID resolution for data in COBOL applications that use the Db2 coprocessor

Variable	ENCODING bind option	COBOL compiler options		CCSID that Db2 uses for the data
		CODEPAGE(nnnn)[1]	(NO)SQLCCSID	
PIC X	not explicitly specified	1140	SQLCCSID	1140[2]
PIC X	not explicitly specified	1140	NOSQLCCSID	Subsystem default application encoding scheme (DECP value APPENSCH)[3]
PIC X	273	1140	SQLCCSID	1140[2]
PIC X	273	1140	NOSQLCCSID	273[4]
PIC X	UNICODE	1140	SQLCCSID	1140[2]
PIC X	UNICODE	1140	NOSQLCCSID	1208 This CCSID does not logically make sense for COBOL. [5]
PIC N USAGE NATIONAL	1140	1140	NOSQLCCSID	1200[6]

Notes:

1. This value can be the value that you explicitly specify with the CODEPAGE compiler option or the default COBOL compiler code page.
2. Because you specified SQLCCSID, Db2 uses the code page value from the COBOL compiler.
3. Because you specified NOSQLCCSID, Db2 does not use the COBOL code page value. Additionally, because you did not explicitly specify a value for the ENCODING bind option, Db2 uses the default application encoding scheme.
4. Because you specified NOSQLCCSID, Db2 does not use the COBOL code page value. Instead, Db2 uses the value that you specified for the ENCODING bind option.
5. Because you specified NOSQLCCSID and the ENCODING bind option UNICODE, Db2 uses CCSID 1208, which is UTF-8. However, COBOL does not support 1208 as a native data type. So, although you can specify this combination of options, do not do so.
6. Because you specified a PIC N USAGE NATIONAL variable, Db2 uses CCSID 1200.

Related tasks

Controlling the CCSID for COBOL host variables (Db2 Application programming and SQL)

Related reference

Compiler options (COBOL) (Enterprise COBOL for z/OS Programming Guide)

Planning to modify compiler option default values (COBOL) (Enterprise COBOL for z/OS Customization Guide)

Using national data (Unicode) in COBOL (Enterprise COBOL for z/OS Programming Guide)

BIND and REBIND options for packages, plans, and services (Db2 Commands)

DECLARE VARIABLE (Db2 SQL)

Subsystem CCSIDs and encoding schemes

Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

Specifying CCSIDs for PL/I applications when using the Db2 coprocessor

If you are using the Db2 coprocessor to prepare a PL/I application with SQL statements, use the PL/I compiler to specify the CCSID of the application source code. For optimal performance, use Db2 to specify the CCSID of the application data in SQL statements.

About this task

The PL/I compiler accepts only one CCSID value that it uses for both the application source code and data. However, Db2 can accept one CCSID value for the source code and one or more CCSID values for the data that is manipulated in SQL statements through host variables and parameter markers.

Procedure

To specify CCSIDs for PL/I applications when using the Db2 coprocessor:

1. To specify the CCSID of the PL/I application source code, use the CODEPAGE compiler option. [4]

   Example: Both of the following JCL EXEC statements for PL/I compile jobs specify a CCSID of 37:

   ```
   //PLI EXEC PGM=IBMZPLI,PARM='...,PP(SQL,...),CODEPAGE(37)...'
   ```

   ```
   //PLI EXEC PGM=IBMZPLI,PARM='...,PP(SQL,...),CP(37)...'
   ```

   Otherwise, if you do not specify the CODEPAGE compiler option, the default PL/I compiler CCSID is passed to the Db2 coprocessor and is used as the CCSID for the source code. The default PL/I compiler CCSID is 1140 unless you changed it.

   Example: The following JCL EXEC statement for a PL/I compile job does not explicitly specify a CCSID. In this case, the PL/I compiler passes the default CCSID, 1140, to the Db2 coprocessor.

   ```
   //PLI EXEC PGM=IBMZPLI,PARM='PP(SQL())'
   ```

   CCSID 1140 is equivalent to CCSID 37 plus the euro symbol (€). However, be aware that conversions, and thus conversion cost, still occur between CCSID 1140 and CCSID 37.

   Recommendation: If you are using the Db2 coprocessor to prepare a PL/I application, do not specify the PL/I preprocessor option SQL with the CCSID suboption. If you specify it anyway, and it conflicts with the CODEPAGE compiler value, Db2 issues a warning. [3] For example, the following EXEC statement for a PL/I compile job specifies the CODEPAGE compiler option with a CCSID value of 1140; the CCSID value 37 is ignored:

   ```
   //PLI EXEC PGM=IBMZPLI,PARM='CP(1140),PP(SQL("CCSID(37)"))'
   ```

   To verify which CCSID value was used, look at the compiler listing and check the CCSID option.
2. To specify whether Db2 or the PL/I compiler determines the CCSID of the application data in SQL statements, specify one of the following PP compiler options:

   CCSID0 (default)
   Specifies that only host variables with the WIDECHAR data type are to be assigned a CCSID value by the Db2 coprocessor (PL/I SQL preprocessor). Host variables with the WIDECHAR data type are always assigned a CCSID value of 1200.

[4] If you are using an older compiler that does not otherwise pass a CCSID value to Db2, use the PL/I preprocessor option 'PP(SQL...' with the CCSID suboption to specify the CCSID of the application source. For example, if you are using an older compiler that does not pass a CCSID value to the Db2 coprocessor, the following EXEC statement for a PL/I compile job specifies a source CCSID of 37.

```
/PLI EXEC PGM=IBMZPLI,PARM='...,PP(SQL("CCSID(37)"),...),...'
```

If your program updates FOR BIT DATA columns with a data type that is not BIT data, choose CCSID0. CCSID0 informs Db2 that the host variable is not associated with a CCSID, allowing the assignment to be made.

Specifying CCSID0 can yield better performance, because you can then use a Db2 mechanism, such as the ENCODING bind option, to specify the host variable CCSIDs.

CCSID0 simulates the behavior of the Db2 precompiler, which does not use the CCSID from the PL/I compiler for host variable data.

Example: The following examples of the PARM clause specify the PP PL/I preprocessor option of the PL/I compiler with the SQL suboption CCSID0

```
PARM='S,XREF,PP(SQL(APOSTSQL,CCSID0))'
PARM='S,XREF,PP(SQL(APOSTSQL))'
```

NOCCSID0

Allows host variables to be assigned a CCSID value by the PL/I SQL preprocessor. If the PL/I SQL preprocessor CODEPAGE option is not also specified, or if the PL/I SQL preprocessor NOCODEPAGE option is also specified, the compiler CODEPAGE option is used as the CCSID for SQL host variables of the CHARACTER data type.

Example: The following example of the PARM clause specifies the PP PL/I preprocessor option of the PL/I compiler with the SQL suboption NOCCSID0

```
PARM='S,XREF,PP(SQL(APOSTSQL,NOCCSID0))'
```

CODEPAGE

Specifies that the compiler CODEPAGE option is always used as the CCSID for SQL host variables of the CHARACTER type.

Example: The following example of the PARM clause specifies the PP PL/I preprocessor option of the PL/I compiler with the SQL suboption CODEPAGE.

```
PARM='S,XREF,PP(SQL(APOSTSQL,CODEPAGE))'
```

3. Optional: If you specified the CCSID0 option, explicitly specify a value for the ENCODING bind option

   The ENCODING bind option value is used as the CCSID for variables in static SQL statements. If you do not explicitly specify a value, the default value for the ENCODING bind option is used.
4. To override the CCSID for particular variables or parameter markers, use the DECLARE VARIABLE statement or CURRENT APPLICATION ENCODING SCHEME special register. If you need help using either of these techniques, following the instructions in Specifying a CCSID for your application.

   If you specify different CCSIDs for different pieces of data, do so with caution.
5. If you want to specify a Unicode CCSID for a particular variable, declare it as a WIDECHAR variable, and use the default SQL preprocessor option of CCSID0. For PL/I variables in applications that are processed by the Db2 coprocessor, the Db2 coprocessor always uses CCSID 1200 for WIDECHAR variables. You do not need to use a DECLARE VARIABLE statement with the CCSID clause for these variables.

Example

The following table shows examples of the CCSID that Db2 uses for data in PL/I applications depending on the options that you specify.

Table 20. CCSID resolution for data in PL/I applications that use the Db2 coprocessor

Variable	ENCODING bind option	PL/I options		CCSID that Db2 uses for the data
		PL/I CODEPAGE compiler option[1]	PP(SQL) suboption	
CHAR(n)	not explicitly specified	1140	NOCCSID0	1140[2]

Table 20. CCSID resolution for data in PL/I applications that use the Db2 coprocessor (continued)

Variable	ENCODING bind option	PL/I options		CCSID that Db2 uses for the data
		PL/I CODEPAGE compiler option[1]	PP(SQL) suboption	
CHAR(n)	not explicitly specified	1140	CCSID0	Subsystem default application encoding scheme (DECP value APPENSCH)[3]
CHAR(n)	500	1140	NOCCSID0	1140[2]
CHAR(n)	500	1140	CCSID0	500[4]
CHAR(n)	UNICODE	1140	NOCCSID0	1140[2]
CHAR(n)	UNICODE	1140	CCSID0	1208 This CCSID and options combination does not logically make sense for PL/I. 5
UCHAR(n)	1208	1140	CCSID0	1208[4]
WIDECHAR(n)	1140	1140	CCSID0	1200[6]

Note:

1. This value can be the value that you explicitly specify with the CODEPAGE compiler option or the default PL/I compiler code page.
2. Because you specified NOCCSID0, Db2 uses the code page value from the PL/I compiler.
3. Because you specified CCSID0, Db2 does not use the PL/I code page value. Additionally, because you did not explicitly specify a value for the ENCODING bind option, Db2 uses the default application encoding scheme.
4. Because you specified CCSID0, Db2 does not use the PL/I code page value. Instead, Db2 uses the value that you specified for the ENCODING bind option.
5. Because you specified CCSID0 and the ENCODING bind option UNICODE, Db2 uses CCSID 1208, which is UTF-8. Although you can specify this combination of options, do not do so. Specify compiler option CODEPAGE(1208) and the NOCCSID0 and CODEPAGE Db2 coprocessor options instead.
6. Because you specified a WIDECHAR variable, Db2 uses CCSID 1200.

Related reference

Compile-time option descriptions (PL/I) (Enterprise PL/I for z/OS Programming Guide:)
Changing the default options (PL/I) (Enterprise PL/I for z/OS Programming Guide:)
SQL preprocessor options (PL/I) (Enterprise PL/I for z/OS Programming Guide:)
BIND and REBIND options for packages, plans, and services (Db2 Commands)
DECLARE VARIABLE (Db2 SQL)
Subsystem CCSIDs and encoding schemes

Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

PL/I PP compiler option

When you specify the CCSID for a PL/I application, you might need to use the PP compiler option. This option enables you to specify SQL processing options to the Db2 coprocessor and the PL/I SQL preprocessor.

The following code shows an example of the PP compiler option:

```
PP(SQL('APOSTSQL,FLOAT(IEEE)'))
```

PL/I has an SQL preprocessor that works with the Db2 coprocessor to process SQL statements. Some of the SQL suboptions for the PP compiler option are for the PL/I SQL preprocessor. Other suboptions are for the Db2 coprocessor. For example, the NOCCSID0 and the CCSID0 suboptions are for the PL/I SQL preprocessor.

The following code shows an example of specifying SQL suboptions for both the Db2 coprocessor and PL/I SQL preprocessor:

```
PP(SQL('APOSTSQL,FLOAT(IEEE),NOCCSID0'))
```

Example: Suppose that you specify the following statement with the PP compiler option and SQL suboption NOCCSID0:

```
PARM='S,XREF,PP(SQL("APOSTSQL,NOCCSID0"))'
```

The following output listing shows the options that are in effect for both the PL/I SQL preprocessor and the Db2 coprocessor:

```
SQL Preprocessor Options Used
NOCCSID0
LOB(DB2)
OPTIONS
DB2 for z/OS Coprocessor Options used
APOST
APOSTSQL
ATTACH(TSO)
CCSID(1140)
```

Specifying CCSIDs for C/C++ applications when using the Db2 coprocessor

If you use the Db2 coprocessor to prepare a C/C++ application with SQL statements, use the C/C++ compiler to specify the CCSID of the application source. Use Db2 mechanisms to specify the CCSID of the application data that is manipulated in SQL statements.

Procedure

To specify CCSIDs for C/C++ applications when using the Db2 coprocessor:

1. Specify the CCSID for the source code by specifying a LOCALE value.

   Example: The following JCL EXEC statements for C compile jobs specify a CCSID of 1047. The first statement specifies a LOCALE for U.S. applications. The second statement specifies a LOCALE for German applications.

   ```
   //C EXEC PGM=CCNDRVR,PARM='SQL(),SO,LIST,LOCALE(En_US.IBM-1047)'
   ```

   ```
   //C EXEC PGM=CCNDRVR,PARM='SQL(),SO,LIST,LOCALE(De_CH.IBM-1047)'
   ```

   Alternatively, you can use other more advanced methods to specify the CCSID of the source code and any data that is outside of SQL statements. For more information about those methods, see the internationalization information in the C/C++ Programming Guide.

Otherwise, if you do not specify a CCSID to the C/C++ compiler, the default C/C++ LOCALE of 1047 is passed to the Db2 coprocessor.

Example: The following JCL EXEC statements for a C compile job does not specify a LOCALE value. Therefore, the default value of 1047 is passed to the Db2 coprocessor.

```
//C EXEC PGM=CCNDRVR,PARM='SQL()'
```

Recommendation: If you are using the Db2 coprocessor on a C/C++ application, do not specify the SQL compiler option with the CCSID suboption. If you specify it anyway, and it conflicts with the LOCALE value, Db2 issues a warning. For example, the following EXEC statement for a C compile job specifies a CCSID value of 1047; the CCSID value 37 is ignored:

```
//C EXEC PGM=CCNDRVR,PARM='SQL(CCSID(37)),LOCALE(De_CH.IBM-1047)'
```

2. Specify the CCSID for data within SQL statements by specifying a value for the ENCODING bind option bind option or accept the subsystem default application encoding scheme (the DECP value APPENSCH).

   This value is used as the CCSID for the application data.
3. To override the CCSID for particular host variables or parameter markers, use the DECLARE VARIABLE statement or CURRENT APPLICATION ENCODING SCHEME special register. If you need help using either of these techniques, following the instructions in Specifying a CCSID for your application.

   If you specify different CCSIDs for different pieces of data, do so with caution.

Related reference

Compiler Options (C/C++) (XL C/C++ User's Guide)

BIND and REBIND options for packages, plans, and services (Db2 Commands)

Subsystem CCSIDs and encoding schemes

Each Db2 subsystem has a set of default CCSID and encoding scheme values. Db2 uses these values for objects and applications that do not otherwise have a CCSID associated with them.

Determining the CCSID of Db2 data

Db2 can store EBCDIC, ASCII, and Unicode data.

Procedure

To determine the CCSID of Db2 data, use one of the following techniques:

- To find the CCSID of data that is stored in Db2 tables, check one of the following catalog tables:
  - SYSIBM.SYSCOLUMNS (the FOREIGNKEY and CCSID columns)

    ```
    SELECT FOREIGNKEY, CCSID
      FROM SYSIBM.SYSCOLUMNS
      WHERE NAME = 'column-name'
    ```

  - SYSIBM.SYSDATABASE (the SBCS_CCSID, MIXED_CCSID, and DBCS_CCSID columns)

    ```
    SELECT SBCS_CCSID, MIXED_CCSID, DBCS_CCSID
      FROM SYSIBM.SYSDATABASE
      WHERE NAME = 'database-name'
    ```

  - SYSIBM.SYSTABLES (the ENCODING_SCHEME column)

    ```
    SELECT ENCODING_SCHEME
      FROM SYSIBM.SYSTABLES
      WHERE NAME = 'table-name'
    ```

  - SYSIBM.SYSTABLESPACE (the SBCS_CCSID, MIXED_CCSID, and DBCS_CCSID columns)

    ```
    SELECT SBCS_CCSID, MIXED_CCSID, DBCS_CCSID
      FROM SYSIBM.SYSTABLESPACE
      WHERE NAME = 'tablespace-name'
    ```

  - SYSIBM.SYSKEYTARGETS (the CCSID column)

```
SELECT CCSID
  FROM SYSIBM.SYSKEYTARGETS
  WHERE IXNAME = 'keytarget-name'
```

- To find the CCSID of a distinct type, check the SYSIBM.SYSDATATYPES catalog table (the ENCODING_SCHEME column).
- To find the CCSID of routine parameters, check one of the following catalog tables:
  - SYSIBM.SYSPARMS (the CCSID column)
  - SYSIBM.SYSROUTINES (the PARAMETER_CCSID column)
- To find the CCSID of application data, check one of the following catalog tables:
  - SYSIBM.SYSPACKAGE (the ENCODING_CCSID column)
  - SYSIBM.SYSPLAN (the ENCODING_CCSID column)
  - SYSIBM.SYSENVIRONMENT (the APPLICATION_ ENCODING_CCSID column)
- To find the subsystem CCSIDs, follow the instructions for "Determining current subsystem CCSID and encoding scheme values" on page 33.

Related reference

Db2 catalog tables (Db2 SQL)

Determining the CCSID of a string value in an SQL statement

Knowing the CCSID of a particular string value in an SQL statement helps you determine how Db2 evaluates the statement. This knowledge also helps you plan for character conversions. You can determine whether character conversion is necessary and what character conversions you need to define.

About this task

The CCSID that is associated with a string value depends on the SQL statement in which the data is referenced and the type of expression.

Procedure

To determine the CCSID of a string value in Db2, use one or more of the following techniques:

- Use the rules for determining the CCSID that is associated with string data, as specified in Determining the encoding scheme and CCSID of a string (Introduction to Db2 for z/OS).
- Use the DESCRIBE statement and then check the SQLDA. The SQLDA contains one SQLVAR entry for each column or host variable that is described. For string columns and parameters, look in the SQLDATA field of the appropriate SQLVAR entry to find the CCSID of that column or parameter.

Related concepts

Code pages and CCSIDs

Because computers store only numbers, they store letters and other characters by assigning a number to them. Which number is mapped to each character depends on the CCSID and code page that is associated with that character.

Related reference

DESCRIBE (Db2 SQL)

SQL descriptor area (SQLDA) (Db2 SQL)

Objects with different CCSIDs in the same SQL statement

You can reference data with different CCSIDs from the same SQL statement. This ability is useful if you use table objects such as tables, views, temporary tables, query tables, and user-defined functions with

different CCSIDs. However, you should understand how Db2 for z/OS processes these queries so that you can code them correctly.

Although the data that the statement references can have different CCSIDs, the SQL statement, including string constants, is written in only one CCSID. The CCSID that the SQL statement is written in is the source CCSID for your application.

Db2 for z/OS considers any SQL statement that satisfies at least one of the following criteria to be a statement that references objects with multiple CCSIDs: GUPI

- References table objects with different CCSIDs
- Contains any of the following functions:
  - ASCII_CHR
  - ASCII_STR
  - ASCIISTR
  - EBCDIC_CHR
  - EBCDIC_STR
  - CAST with the CCSID clause
  - CHR
  - DECRYPT_BIT
  - DECRYPT_CHAR
  - DECRYPT_DB
  - GETVARIABLE
  - GX
  - NORMALIZE_STRING
  - UNICODE_STR
  - UNISTR
  - UX
  - XML2CLOB
  - XMLSERIALIZE
  - XMLTABLE
  - A table user-defined function
- Is one of the following SQL statements:
  - CALL
  - SET host-variable assignment
  - SET *special register*
  - VALUES
  - VALUES INTO

GUPI

If a statement references objects with multiple CCSIDs, Db2 processes the statement as follows:

1. Db2 first determines the CCSID for each item that the statement references. Db2 uses the rules in the table that describes the operand types in Conversion rules for comparisons (Db2 SQL).
2. Db2 then evaluates the predicates according to the rules that are listed in the "Operand that supplies the CCSID for character conversion" table in Conversion rules for comparisons (Db2 SQL).

Regardless of the CCSIDs of the referenced data, your application can receive the data in any CCSID that it wants. For example, suppose that your application selects rows from SYSIBM.SYSTABLES. The CCSIDs of the retrieved data are all Unicode CCSIDs. However, when you issue the SELECT statement, the data is

returned to your application in your application encoding CCSID. This behavior is evident in the SPUFI application, which uses the EBCDIC encoding scheme. When you run a query against the Db2 catalog in SPUFI, EBCDIC data is returned.

Examples of statements that reference objects with different CCSIDs

GUPI

Example 1: Assume that EBCDICTABLE is encoded in EBCDIC, and the host variables are encoded in the application encoding scheme. SYSIBM.SYSTABLES is encoded in Unicode. Consider the following statement that references these objects with different CCSIDs:

```
SELECT A.NAME, A.CREATOR, B.CHARCOL, 'ABC', :hvchar, X'C1C2C3'
FROM SYSIBM.SYSTABLES A, EBCDICTABLE B
WHERE A.NAME = B.NAME AND
      B.NAME > 'B' AND
      A.CREATOR = 'SYSADM'
ORDER BY B.NAME
```

Db2 uses the following CCSIDs for each item that the statement references:

Part of statement	Corresponding CCSID that Db2 uses during evaluation of the statement
A.NAME	Unicode CCSID
A.CREATOR	Unicode CCSID
B.CHARCOL	EBCDIC CCSID
'ABC'	Application encoding scheme CCSID[1]
:hvchar,	Application encoding scheme CCSID[1]
X'C1C2C3'	Application encoding scheme CCSID[1]
B.NAME	EBCDIC

Notes:

1. Application encoding scheme CCSID is the value of the ENCODING bind option.

Db2 then evaluates the statement as follows:

Part of statement	Corresponding CCSID that Db2 uses during evaluation of the statement	Reason
A.NAME = B.NAME	Unicode CCSID	Because both operands are columns and the CCSIDs are different, Db2 uses Unicode.
B.NAME > 'B'	EBCDIC CCSID	Because the first operand is a column and the second operand is a string, Db2 uses the CCSID of the first operand, which is EBCDIC.
A.CREATOR = 'SYSADM'	Unicode CCSID	Because the first operand is a column and the second operand is a string, Db2 uses the CCSID of the first operand, which is Unicode.

The result of this statement contains multiple CCSIDs. However, your application receives the result of this statement in the application encoding CCSID.

Example 2: Assume that you issue the following statements to create and populate a Unicode table and EBCDIC table:

```
CREATE TABLE TCCSIDU (CU1 VARCHAR(12)) CCSID UNICODE;
CREATE TABLE TCCSIDE (CE1 VARCHAR(12)) CCSID EBCDIC;
INSERT INTO TCCSIDU VALUES (‘Jürgen');
INSERT INTO TCCSIDE VALUES ('Jürgen');
```

The following query joins those two tables.

```
SELECT LENGTH(A.CU1) AS L1, HEX(A.CU1) AS H1,
LENGTH(B.CE1) AS L2, HEX(B.CE1) AS H2
FROM TCCSIDU A, TCCSIDE B WHERE A.CU1 = B.CE1;
```

The WHERE predicate compares two columns with different CCSIDs. Column A.CU1 is encoded in Unicode. Column B.CE1 is encoded in EBCDIC. For this comparison, Db2 promotes B.CE1 to Unicode. Therefore Db2 evaluates the EBCDIC value 'Jürgen' in B.CE1 as equal to the Unicode value 'Jürgen' in A.CU1. This query returns the following result:

L1	H1	L2	H2
7	4AC3BC7267656E	6	D1DC99878595

Even though B.CE1 was promoted to Unicode for the comparison in the WHERE clause, the result still shows the EBCDIC hexadecimal value for B.CE1.

GUPI

Related tasks

Specifying a CCSID for your application

In Db2 for z/OS applications, one CCSID is associated with the source code and one or more CCSIDs can be associated with the data that your application manipulates. The CCSID that Db2 associates with the data is called the *application encoding scheme*.

Related reference

DESCRIBE (Db2 SQL)

SQL descriptor area (SQLDA) (Db2 SQL)

Differences between Unicode and EBCDIC sorting sequences

In Unicode, numeric characters are sorted before alphabetic characters. In EBCDIC, alphabetic characters are sorted before numeric characters.

Because the Db2 catalog is stored in Unicode, any queries that you issue against Unicode tables in the catalog use the Unicode sorting sequence.

Also, consider any SQL statements that include syntax that requires that the data be sorted. Examples of such syntax include the GROUP BY clause, range predicates such as BETWEEN, and functions such as MIN and MAX. These statements might return different results when they are issued on Unicode data than on EBCDIC data.

The following table shows some example encoding differences to consider when specifying these clauses, predicates, and functions in your SQL statements.

Table 21. Example encoding differences

EBCDIC		Unicode and ASCII	
Characters	**Hexadecimal value**	**Characters**	**Hexadecimal value**
space	X'40'	space	X'20'

Table 21. Example encoding differences (continued)

EBCDIC		Unicode and ASCII	
Characters	**Hexadecimal value**	**Characters**	**Hexadecimal value**
lowercase characters	X'81' - X'89' X'91' - X'99' X'A1' - X'A9'	numerals	X'30' - X'39'
uppercase characters	X'C1' - X'C9' X'D1' - X'D9' X'E1' - X'E9'	uppercase characters	X'40' - X'4F' X'50' - X5A'
numerals	X'F0' - X'F9'	lowercase characters	X'61' - X'6F' X'70' - X7A'

Equal predicates are not affected by the different sorting sequences.

Examples

GUPI

For the following examples, assume that a table called MYTABLES has a NAME column that is type VARCHAR(128). This column contains the following values: TEST1, TEST2, TEST3, TESTA, TESTB, and TESTC.

Example query with ORDER BY: Suppose that you issue the following SQL query:

```
SELECT NAME FROM MYTABLES
ORDER BY NAME
```

If MYTABLES is encoded in Unicode, Db2 returns the following result:

```
TEST1
TEST2
TEST3
TESTA
TESTB
TESTC
```

If MYTABLES is encoded in EBCDIC, Db2 returns the following result:

```
TESTA
TESTB
TESTC
TEST1
TEST2
TEST3
```

Example of query with ORDER BY and BETWEEN predicate: Assume that you issue the following SQL query:

```
SELECT * FROM MYTABLES
WHERE NAME BETWEEN 'TEST2' AND 'TESTB'
ORDER BY NAME
```

If MYTABLES is encoded in Unicode, Db2 returns the following result:

```
TEST3
TESTA
```

If MYTABLES is encoded in EBCDIC, Db2 returns the following result:

```
TESTC
TEST1
```

To simulate the behavior of the ORDER BY clause on EBCDIC data, use the CAST function and the ORDER BY clause when you query the Db2 catalog or other Unicode data.

Example of simulating the EBCDIC sorting sequence: Suppose that MYTABLES is encoded in Unicode. You can modify the preceding query as follows to return the Unicode data in the same order that you would expect for EBCDIC data:

```
SELECT CAST(NAME AS VARCHAR(128) CCSID EBCDIC) AS E_NAME
FROM MYTABLES
ORDER BY E_NAME
```

Db2 returns the following result:

```
TESTA
TESTB
TESTC
TEST1
TEST2
TEST3
```

However, be aware that, in this situation, Db2 cannot use an index to support the ORDER BY clause. Db2 must sort the data.

You can also apply this same technique to a catalog table in UTF-8, as shown in the following example

```
SELECT CAST(NAME AS VARCHAR(128) CCSID EBCDIC) AS E_NAME
FROM SYSIBM.SYSTABLES
ORDER BY E_NAME
```

If the NAMES column of SYSIBM.SYSTABLES contains the values TEST1, TEST2, TEST3, TESTA, TESTB, and TESTC, Db2 returns the following result:

```
TESTA
TESTB
TESTC
TEST1
TEST2
TEST3
```

GUPI

Specifying how Db2 calculates the length of a string

If you use certain length functions, you can specify whether you want Db2 to calculate the length by bytes or characters. This distinction is important for multibyte characters. If you convert Db2 data to Unicode and the data expands, consider updating some of these function calls to specify the appropriate unit of measurement.

For example, consider the string Jürgen in UTF-8. This string consists of 6 characters. However, it takes 7 bytes of storage, because the character ü takes 2 bytes in UTF-8. You can specify whether you want Db2 to count the length as 6 or 7.

The key is to specify the size code unit that you want Db2 to use when calculating the length. A *code unit* is the minimal bit combination that can represent a character.

Procedure

If you are using any of the following length functions, specify the appropriate unit of measurement:

GUPI **Applicable functions:**

- CHARACTER_LENGTH

- CLOB
- DBCLOB
- GRAPHIC
- LEFT
- LOCATE
- LOCATE_IN_STRING
- OVERLAY
- POSITION
- RIGHT
- SUBSTRING
- VARCHAR
- VARGRAPHIC

GUPI

Options to specify unit of measurement:

CODEUNITS16
Specifies that Db2 is to count the length by 16-bit (or 2-byte) code units. For every character that is 2 bytes or less in the string, Db2 counts a length of 1.

CODEUNITS32
Specifies that Db2 is to count the length by 32-bit (or 4-byte) code units. For every character that is 4 bytes or less in the string, Db2 counts a length of 1. CODEUNITS32 always returns the same value as CODEUNITS16 unless you have supplementary characters.

OCTETS
Specifies that Db2 is to count the length by bytes. For every byte in the string, Db2 counts a length of 1.

The OCTETS option is not available for all of the listed functions.

Example

GUPI

Example of CHARACTER_LENGTH: Assume that NAME is a VARCHAR(128) column that is encoded in Unicode UTF-8 and contains 'Jürgen'. The character ü requires two bytes in UTF-8.

The following two queries both return the value 6, because Db2 counts the string Jürgen as 6 characters. In the first query, CODEUNITS32 means that any character that is 4 bytes or less is counted as 1. In the second query, CODEUNITS16 means that any character that is 2 bytes or less is counted as 1. In both cases, the result is the same.

```
SELECT CHARACTER_LENGTH(NAME,CODEUNITS32)
        FROM T1 WHERE NAME = 'Jürgen';
```

```
SELECT CHARACTER_LENGTH(NAME,CODEUNITS16)
        FROM T1 WHERE NAME = 'Jürgen';
```

However the following two queries return the value 7, because the string contains 7 bytes. In the first query, OCTETS means that length is to be calculated in bytes. In the second query, the LENGTH function always counts by bytes.

```
SELECT CHARACTER_LENGTH(NAME,OCTETS)
        FROM T1 WHERE NAME = 'Jürgen';
```

```
SELECT LENGTH(NAME)
        FROM T1 WHERE NAME = 'Jürgen';
```

Example of LOCATE_IN_STRING: The LOCATE_IN_STRING function returns the position at which an occurrence of an argument starts within a specified string. This function is similar to POSITION, but adds a parameter to specify which instance of the search string to find. The following statement sets the value of the host variable POSITION to 26, because the character ß is the 26th character in the string. In this case, CODEUNITS32 means that any character that is 4 bytes or less is counted as 1.

```
SET :POSITION = LOCATE_IN_STRING('Jürgen lives on Hegelstraße','ß',-1,CODEUNITS32);
-- search from end
```

The following statement sets the value of the host variable POSITION to 6. Db2 starts at position 1 and looks for the third occurrence of the character N. In this case, OCTETS means that Db2 counts the length by bytes.

```
SET :POSITION = LOCATE_IN_STRING('WINNING','N',1,3,OCTETS);
```

Examples of other length functions: The following table shows examples of the CODEUNITS16, CODEUNITS32, and OCTET options.

Table 22. Examples of length functions

Function	Result	Hexadecimal result value
`LEFT('Jürgen',2,CODEUNITS32)`	`'Jü'`	X'4AC3BC'
`LEFT('Jürgen',2,CODEUNITS16)`	`'Jü'`	X'4AC3BC'
`LEFT('Jürgen',2,OCTETS)`	`'J '`	X'4A20' (a truncated string)
`LEFT('Jürgen',2)`	`'J?'`	X'4AC3' (The letter 'J' and a partial character)[1]
`RIGHT('Jürgen',5,CODEUNITS32)`	`'ürgen'`	X'C3BC7267656E'
`RIGHT('Jürgen',5,CODEUNITS16)`	`'ürgen'`	X'C3BC7267656E'
`RIGHT('Jürgen',5,OCTETS)`	`'rgen'`	X'207267656E' (a truncated string)
`RIGHT('Jürgen',5)`	`'?rgen'`	X'BC7267656E' (a partial character followed by 'rgen')[1]
`SUBSTRING('Jürgen',1,2,CODEUNITS32)`	`'Jü'`	X'4AC3BC'
`SUBSTRING('Jürgen',1,2,CODEUNITS16)`	`'Jü'`	X'4AC3BC'
`SUBSTRING('Jürgen',1,2,OCTETS)`	`'J '`	X'4A20' (a truncated string)
`SUBSTR('Jürgen',1,2)`	`'J?'`	X'4AC3' (a partial character)
`SUBSTRING('Jürgen',8,CODEUNITS16)`	`''`	a zero-length string
`SUBSTRING('Jürgen',8,4,OCTETS)`	`''`	a zero-length string

Table 22. Examples of length functions (continued)

Function	Result	Hexadecimal result value
1. If conversion occurs on a string with a partial character, SQLCODE -330 results.		

GUPI

Related concepts

Difference between CODEUNITS16 and CODEUNITS32 (Db2 SQL)

String unit specifications (Db2 SQL)

Related reference

CHARACTER_LENGTH (Db2 SQL)

LEFT (Db2 SQL)

LENGTH (Db2 SQL)

LOCATE_IN_STRING (Db2 SQL)

RIGHT (Db2 SQL)

SUBSTR (Db2 SQL)

SUBSTRING (Db2 SQL)

Specifying the sorting sequence for a language

If your application sorts non-English data, you should specify the sorting sequence to ensure that Db2 sorts the data in a culturally correct manner. For example, suppose your data contains the following strings: cote, coté, côte, côté. You need to specify how you want these strings sorted.

Procedure

To specify the sorting sequence for a language, perform one of the following actions:

- In your SQL statement, use the COLLATION_KEY function with the *collation-name* parameter to specify a particular sorting sequence.

  A collation name specifies how Db2 is to sort data. It specifies attributes such as the language of the data, whether case should be considered, and how punctuation characters should be treated. You must specify a value that is acceptable for the z/OS CUNBOPR_Collation_Keyword parameter.

  The COLLATION_KEY function returns a binary value that can be used to sort data according to the rules that are specified in the Unicode Collation algorithm.

  Example of retrieving data in a specified order: GUPI Suppose that you issue the following query:

```
SELECT FIRSTNAME, LASTNAME
FROM EMPLOYEE
ORDER BY COLLATION_KEY(LASTNAME, 'UCA400R1_AS_LSV_S2');
```

  GUPI This query orders the employees by their surnames (in the LASTNAME column) based on the following options that are specified in the collation name UCA400R1_AS_LSV_S2:

Table 23. Example collation options and corresponding collation keywords

Corresponding collation keyword	Option
UCA400R1	Use Unicode Collation Algorithm (UCA) version 4.0.1
AS	Ignore spaces, punctuation and symbols
LSV	Use Swedish linguistic conventions

Table 23. Example collation options and corresponding collation keywords (continued)

Corresponding collation keyword	Option
S2	Compare case-insensitively

- Create an index that maintains the sorting sequence by using the COLLATION_KEY function in the CREATE INDEX statement.

  Invoking the COLLATION_KEY function for every row in the table can slow performance. Creating an index based on the collation key shifts this performance cost from query time to insert or update time. That performance shift assumes that Db2 chooses to use the index for the query.

  Example of creating an index based on the collation key: GUPI Suppose that you want to use the following basic query:

```
SELECT C1 FROM T1 ORDER BY C1
```

  However, you want to ensure that the result is ordered according to the rules for a particular locale. For this example, assume the language of the data is French. In this case, you can use the COLLATION_KEY function, as shown in the following statement:

```
SELECT C1 FROM T1 ORDER BY COLLATION_KEY(C1,'UCA410_LFR_FO')
```

  GUPI The collation name UCA410_LFR_FO has the following meaning:

Table 24. Example collation options and corresponding collation keywords

Corresponding collation keyword	Option
UCA410	Specifies that Db2 is to use the collation service UCA410
LFR	Specifies that the locale is French. (L = language, FR= French)
FO	Specifies that the French sorting attribute is to be used. (F = French attribute, O = On) Strings are to be sorted by examining the accents starting from the end of the string. This attribute is automatically set to on for the French locales. Therefore, in this case, it is not required.

  You might want to check if you can improve the performance of this query by creating an index on C1 that is based on the collation key. The following example statements show how to create such an index and use EXPLAIN statements to confirm that the index is used for faster access. You can view the results of the EXPLAIN statements by querying the plan table. The EXPLAIN output for this example shows only some of the plan table columns.

  PSPI

```
EXPLAIN ALL SET QUERYNO = 110 FOR SELECT C1 FROM T1
   ORDER BY COLLATION_KEY(C1,'UCA410_LFR_FO');
CREATE INDEX I1 ON T1 (COLLATION_KEY(C1,'UCA410_LFR_FO'));
EXPLAIN ALL SET QUERYNO = 210 FOR SELECT C1 FROM T1
   ORDER BY COLLATION_KEY(C1,'UCA410_LFR_FO');
SELECT * FROM PLAN_TABLE;
```

  PSPI The last statement returns the following output:

```
+-----------------------------------------------------------------------------------------
     | QUERYNO | QBLOCKNO | PROGNAME | PLANNO | METHOD | CREATOR | TNAME | TABNO |
ACCESSTYPE |
```

```
+-------------------------------------------------------------------------------------------
   1_|     110 |        1 | DSNTEP2  |      1 |      0 | ADMF001 | T1    |     1 |
R          |
   2_|     110 |        1 | DSNTEP2  |      2 |      3 |         |       |     0
|            |
   3_|     210 |        1 | DSNTEP2  |      1 |      0 | ADMF001 | T1    |     1 |
I          |

+-------------------------------------------------------------------------------------------

     ------------------------------------------------------------------------------
     | MATCHCOLS | ACCESSNAME | INDEXONLY | SORTN_UNIQ | SORTN_JOIN | SORTN_ORDERBY |
     ------------------------------------------------------------------------------
   1_|         0 |            | N         | N          | N          | N             |
   2_|         0 |            | N         | N          | N          | N             |
   3_|         0 | I1         | N         | N          | N          | N             |
     ------------------------------------------------------------------------------

     ---------------------------------------------------------------------------------------------
     | SORTN_GROUPBY | SORTC_UNIQ | SORTC_JOIN | SORTC_ORDERBY | SORTC_GROUPBY | PREFETCH   |

-------------------------------------------------------------------------------------------
   1_| N             | N          | N          | N             | N             |  S         |
   2_| N             | N          | N          | Y             | N             |            |
   3_| N             | N          | N          | N             | N             |            |
     ---------------------------------------------------------------------------------------------
```

Related reference

COLLATION_KEY (Db2 SQL)

CREATE INDEX (Db2 SQL)

EXPLAIN (Db2 SQL)

Related information

Unicode Technical Standard #10: Unicode Collation Algorithm

Performing culturally correct case conversions

Rules for uppercase and lowercase conversion vary according to language and country. If you plan to use the UPPER or LOWER function, you need to ensure that Db2 uses the culturally correct casing rules. For example, you need to tell Db2 how to convert characters such as ß and ó to uppercase.

Before you begin

Before you use the UPPER or LOWER function on Unicode or ASCII data, you need to set up z/OS Unicode Services.

Procedure

To ensure that Db2 uses the correct casing rules for a language and country:

- When you use the UPPER function or LOWER function, ensure that Db2 uses the appropriate locale by performing one of the following actions:
  - Specify a value for the *locale-name* parameter of the UPPER or LOWER function:
    - For EBCDIC data, specify an LE locale, such as En_US or Fr_FR.
    - For Unicode and ASCII data, specify a locale value that is supported by the case conversion service of z/OS Unicode Services, such as EN_US. For a list of locale values that are supported by the case conversion service, see Locales supported for case service (z/OS: Unicode Services User's Guide and Reference). You can also specify the value UNI, which means that the case conversion service of z/OS Unicode Services is to use the normal and special casing rules.
  - If you do not specify a value for the *locale-name* parameter of the UPPER and LOWER function, ensure that the value of the CURRENT LOCALE LC_CTYPE special register is correct. You can change the value by using the SET CURRENT LOCALE LC_CTYPE statement.

  As an alternative to using the UPPER function, you can use the TRANSLATE function with only one parameter. In both cases, the strings are converted to uppercase.

Example

GUPI

Example 1

The following statements show how to ensure that the German character ß is handled correctly when Db2 converts it to upper case. In uppercase, ß should be 'SS.'

The first set of statements creates a table, inserts one row, and confirms that the value Hegelstraße was properly inserted.

```
CREATE TABLE T1 (C1 VARCHAR(15)) VOLATILE CCSID UNICODE;
INSERT INTO T1 VALUES('Hegelstraße',1);
SELECT C1 FROM T1 ;
```

The SELECT statement returns the following result:

C1
Hegelstraße

If you do not specify a locale when you use the UPPER function on this value, the result is technically incorrect, as shown in the following example. In upper case, the German ß should be converted to SS.

```
SELECT UPPER(C1)AS C1 FROM T1 ;
```

This SELECT statement returns the following result:

C1
HEGELSTRAßE

The following query returns output with the German ß correctly converted to SS.

```
SELECT UPPER(C1,'De_DE') AS C1 FROM T1 ;
```

This SELECT statement returns the following result:

C1
HEGELSTRASSE

This query works correctly, because the locale De_DE is passed as a parameter to the UPPER function.

Example 2

Suppose that table T1 contains the Unicode data Chrysóstomo in column C1. Assume that you issue the following query with the UPPER function.

```
SELECT UPPER(C1)AS C1 FROM T1 ;
```

If you did not add the CASE SPECIAL and CASE LOCALE statements to your conversion image when setting up z/OS Unicode Services, this query returns the following result:

```
CHRYSóSTOMO
```

However, after setting up the conversion image with the CASE SPECIAL and CASE LOCALE statements and setting the LOCALE special register, you get the following correct result:

```
CHRYSÓSTOMO
```

Be aware that the UPPER function can result in expansion if the text contains certain characters, such as ó in this example. Ensure that the result string is large enough to contain the result of the expression.

GUPI

Related reference

LOWER (Db2 SQL)
UPPER (Db2 SQL)
TRANSLATE (Db2 SQL)
CURRENT LOCALE LC_CTYPE (Db2 SQL)
Internationalization: Locales and Character Sets (XL C/C++ Programming Guide)
Locales for collation and case support (z/OS: Unicode Services User's Guide and Reference)

Locale

A *locale* defines your cultural environment. Specifying the correct locale ensures that Db2 handles case conversions and sorts according to the rules for a particular language.

You can set the locale for your subsystem by using the CURRENT LOCALE LC_CTYPE special register. Alternatively, you can specify a locale when you perform specific functions that depend on locale, such as UPPER and LOWER.

Depending on the encoding scheme of the data, use one of the following locale formats:

LE locales

Specify this locale format for EBCDIC data.

An LE locale consists of two components: the first component represents a specific language and country, and the second component is a CCSID. For example, in the locale Fr_CA.IBM-1047, Fr_CA represents the language and country (French Canadian), and IBM-1047 is the associated CCSID.

When you specify an LE locale to Db2 for z/OS, specify only the first component, which is the language and country. Db2 appends "IBM-" and the CCSID.

The following table shows some example LE locales that you can specify to Db2.

Table 25. Examples of LE locales that you can specify

Locale	Language	Country
En_US	English	United States
De_CH	German	Switzerland
De_DE	German	Germany
Fr_CA	French	Canada
It_IT	Italian	Italy
Ja_JP	Japanese	Japan

For a complete list of supported LE locales, see Compiled locales (LE locales) (XL C/C++ Programming Guide).

z/OS Unicode Services

Specify one of the following z/OS Unicode Services locale formats for ASCII and Unicode data.

Locale format for case conversion services

Use this format when you need a locale to affect how data is converted to uppercase and lowercase, such as in the UPPER and LOWER functions.

The locale format for case conversion is L*xx*_R*yy* where:

L*xx*
Language

R*yy*
Region

You can use any of the locale values that are supported by z/OS Unicode Services for CUNBAPRM_Locale (31-bit) or CUN4BAPR_Locale (64-bit). The following table lists some example locale values for case conversion services.

Table 26. Example locale values for the case conversion services of z/OS Unicode Services

Locale value	Language	Region
Cs_CZ	Czech	Czech Republic
De_DE	German	Germany
En_US	English	United States
En_GB	English	Great Britain
Es_MX	Spanish	Mexico
Fr_FR	French	France
Ja_JP	Japanese	Japan
Sv_SE	Swedish	Sweden

For a complete list of supported locales for case conversion services, see Locales supported for case service (z/OS: Unicode Services User's Guide and Reference).

Locale format for collation conversion services
Use this format when you need a locale to affect how the data is sorted.

The collation locale format is L*xx*_R*yy*_V*zz* where:

L*xx*
Language

R*yy*
Region

V*zz*
Variant

You can use any of the locale values that are supported by z/OS Unicode Services for CUNBOPRM_Collation_Keyword/CUN4BOPR_Collation_Keyword The following table lists some example locale values for collation

Table 27. Example locale values for collation conversions in z/OS Unicode Services

Locale value	Language	Region	Variant
LCS_RCZ	Czech	Czech Republic	None
LDE_RDE	German	Germany	None
LDE_RDE_PREEURO	German	Germany	Pre Euro support
LEN	English	None	None
LEN_RGB	English	Great Britain	None
LES_RMX	Spanish	Mexico	None
LFR_RFR	French	France	None
LJA	Japanese	None	None

Table 27. Example locale values for collation conversions in z/OS Unicode Services (continued)

Locale value	Language	Region	Variant
LSV	Swedish	None	None

For a complete list of supported locales for collation conversion services, see Locales supported for collation (z/OS: Unicode Services User's Guide and Reference).

Generating escaped Unicode data

If you pass Unicode characters to an application or object that is not intended to handle Unicode data, data might be lost unless you escape certain characters. For example, you might need to pass Unicode data through an application that has EBCDIC host variables. Or you might want to store Unicode data in a non-Unicode table.

About this task

You might also want to select Unicode characters from an application that runs on a 3270 terminal emulator, such as SPUFI. If the CCSID setting of the emulator does not include those Unicode characters, those characters do not display properly in the output.

In these situations, those Unicode characters that cannot be represented in the encoding scheme of the application or object are lost unless you escape them. *Escaped data* is one or more characters that cannot be represented in the target CCSID and is instead represented by the encoding value. This representation preserves the data. For example, the escaped version of the Unicode character д is \0434. Thus, the following ASCII string contains the escaped character д: `'The escaped character is \0434'`

If you insert escaped data into a Unicode table, Db2 does not interpret your data and modify it to be un-escaped. Escaped data is stored as is in a Db2 table, regardless of whether the table is an ASCII, EBCDIC, or Unicode table.

Procedure

To generate escaped Unicode data:

1. Use the ASCII_STR function or the EBCDIC_STR function.

   These functions convert a Unicode string to an ASCII or EBCDIC string. Characters that do not exist in ASCII or EBCDIC are converted to the form *xxxx*, where *xxxx* represents a UTF-16 code unit.

   For more information about how to convert characters to UTF-16 format, see step 2 under the instructions for the INSERT statement in "Inserting data into a Unicode table" on page 48.
2. If you later need to convert the EBCDIC or ASCII string with escaped data back to Unicode, use the UNICODE_STR function.

   The short form of the function name is UNISTR. This function interprets escaped data in the source string. Values that are preceded by a backslash ('\') are treated as Unicode UTF-16 characters. For example '\0041' is the Unicode UTF-16 representation for 'A'.

Example

GUPI

Example of escaping data: Assume that T1.C1 contains 'Hi, my name is Андрей'. Notice that the characters in Андрей are all Cyrillic characters, even though some of them do resemble Latin characters. Suppose that you issue the following query in SPUFI:

```
SELECT C1 FROM T1;
```

The result of this query is displayed as follows on a 3270 terminal emulator with the CCSID set to 37:

```
‘Hi, my name is ......'
```

Because the characters in Андрей do not exist in CCSID 37, this name is instead displayed as `.....`. To solve this problem, you can add the EBCDIC_STR function, as shown in the following example:

```
SELECT EBCDIC_STR(C1)FROM T1;
```

Db2 returns the following output with escaped data:

```
‘Hi, my name is \0410\043D\0434\0440\0435\0439'
```

Notice that 0410 is the UTF-16 value for А, 043D is the UTF-16 value for н and so on.

Example of un-escaping data: Assume that T1.C1 contains 'Андрей'. Suppose that you issue the following query:

```
SELECT HEX(UNISTR(ASCII_STR(C1))) FROM T1;
```

Db2 interprets this query as follows:

Table 28. How Db2 interprets query with UNISTR

Part of SELECT statements	Result	Explanation
ASCII_STR(C1)	\0410\043D \0434\0440\0435\0439	Db2 returns the value in C1 (Андрей) as an ASCII string. Because these characters cannot be represented in ASCII, they are escaped.
UNISTR(ASCII_STR(C1))	Андрей	Db2 then converts the escaped ASCII string to a Unicode UTF-8 string. UTF-8 includes all of the characters, so they no longer have to be escaped.
HEX(UNISTR(ASCII_STR(C1)))	D090D0BDD0B4D180D0B5D0B9	Db2 then returns the hexadecimal value of the UTF-8 string.

Thus, the final result of this query is:

```
D090D0BDD0B4D180D0B5D0B9
```

Suppose that you issue the following similar query:

```
SELECT HEX(UNISTR(ASCII_STR(C1),UTF16)) FROM T1;
```

Db2 interprets this query as follows:

Table 29. How Db2 interprets query with UNISTR and UTF16 parameter

Part of SELECT statements	Result	Explanation
ASCII_STR(C1)	\0410\043D \0434\0440\0435\0439	Db2 returns the value in C1 (Андрей) as an ASCII string. Because these characters cannot be represented in ASCII, they are escaped.

Table 29. How Db2 interprets query with UNISTR and UTF16 parameter (continued)

Part of SELECT statements	Result	Explanation
`UNISTR(ASCII_STR(C1),UTF16)`	Андрей	Db2 then converts the escaped ASCII string to a Unicode UTF-16 string. UTF_16 includes all of the characters, so they no longer have to be escaped.
`HEX(UNISTR(ASCII_STR(C1)))`	`D090D0BDD0B4D180D0B5D0B9`	Db2 then returns the hexadecimal value of the UTF-16 string.

Thus, the final result of this query is:

```
0410043D0434044004350439
```

GUPI

Related concepts

Situations in which character conversion occurs

Character conversion is the process of converting data from one CCSID to another CCSID. This process can occur when data is transferred between a remote and local system or when data is manipulated within the local system.

Related tasks

Inserting Unicode data into a non-Unicode table

If you insert Unicode data into an EBCDIC or ASCII table, use escaped data for those characters that cannot be represented in the target encoding scheme. Using escaped data ensures that those characters are preserved.

Related reference

ASCII_STR or ASCIISTR (Db2 SQL)

EBCDIC_STR (Db2 SQL)

HEX (Db2 SQL)

UNICODE_STR or UNICODE_STR (Db2 SQL)

Normalization of Unicode strings

Your application should treat as equal those characters that are functionally and visually equivalent but have different code point representations. This behavior is important when you search, sort, or compare Unicode strings. To accomplish this goal, you might need to normalize the strings. However, normalization can degrade performance.

Unicode strings can be canonically equivalent or compatibly equivalent. If they are canonically equivalent, they are also compatibly equivalent.

Canonically equivalent characters are those characters that are equivalent both functionally and visually, but might have different code point representations. To users, these characters are indistinguishable in that they are displayed identically. For example, the character ü is canonically equivalent to the sequence u and ¨.

Compatibly equivalent characters are characters with plain text that is equivalent, regardless of the semantic meaning. These characters might also have different code point representations. For example, superscript and subscript numerals are compatibly equivalent to their decimal-digit counterparts.

The process of normalization of Unicode strings produces a unique code point sequence for all sequences that are equivalent, either canonically or compatibly. Therefore, all canonically equivalent characters have the same binary representation. You can normalize a Unicode string into one of the following normalized forms that are defined by the Unicode Standard:

Normalization Form Canonical Decomposition (NFD)
Characters are decomposed by canonical equivalence.

Normalization Form Canonical Composition (NFC)
Characters are decomposed and then recomposed by canonical equivalence.

Normalization Form Compatibly Decomposition (NFKD)
Characters are decomposed by compatibly equivalence.

Normalization Form Compatibly Composition (NFKC)
Characters are decomposed by compatibly equivalence and then recomposed by canonical equivalence.

To normalize Unicode strings, use the NORMALIZE_STRING built-in function.

Related reference
NORMALIZE_STRING (Db2 SQL)
Related information
Canonical Equivalence in Applications (Unicode Consortium)

How Db2 handles Unicode supplementary characters

Unicode supplementary characters are those characters that have a code point between U+10000 and U +10FFFF. These characters include certain math symbols and certain characters from Chinese, Japanese, and some historic scripts.

Supplementary characters are also known as surrogate characters. Each one of these characters takes up 4 bytes in either UTF-8 and UTF-16. In UTF-8, each one of these characters takes up four 8-bit code units. In UTF-16, each one of these characters takes up two 16-bit code units.

Db2 detects any supplementary data that is not well formed only if Db2 has to manipulate the data in some way. For example, if Db2 converts the data or processes it as part of a built-in function, Db2 can detect if it is not well formed. Any built-in function that has the CODEUNITS32, CODEUNITS16, and OCTETS options, such as CHARACTER_LENGTH and LOCATE_IN_STRING, can detect whether supplementary characters are well formed. Other operations are also "character aware." For example, LIKE predicates, the truncation of host variables, and character conversion operations need to know the content of any character data.

However, suppose that you insert data into a column and Db2 does not need to manipulate it in any way. In this case, Db2 does not detect problems with data that is not well formed. For example, if your COBOL application, which uses UTF-16 data, inserts garbage data into a GRAPHIC column, Db2 does not report any problems. You can use the NORMALIZE_STRING function to process data and ensure that it is well-formed according to one of the Unicode standard forms. However, using this function might degrade performance.

Related concepts
String unit specifications (Db2 SQL)
Related reference
CHARACTER_LENGTH (Db2 SQL)
LOCATE_IN_STRING (Db2 SQL)
NORMALIZE_STRING (Db2 SQL)
Related information
Unicode Consortium

Processing Unicode data in COBOL applications

COBOL and Db2 for z/OS support UTF-16 data and UTF-8 data.

Procedure

To process Unicode data in COBOL applications for Db2 for z/OS, perform the following recommended actions:

- For Unicode UTF-16 data, use one of the national data types. For example, specify PIC N(10) USAGE NATIONAL. For Unicode UTF-8 data, use the BYTE-LENGTH phrase of the PICTURE clause and the UTF-8 phrase of the USAGE clause. For example, specify PIC U BYTE-LENGTH 10 USAGE UTF-8.

  If you use a COBOL application to retrieve UTF-8 data from Db2 into a variable other than a UTF-8 variable, Db2 converts the output to the format that is required by the application. For example, if you query the Db2 catalog, Db2 converts the data for the COBOL application from UTF-8 to either UTF-16 (for PIC N USAGE NATIONAL variables) or EBCDIC (for PIC X variables). However, you should not store unconverted UTF-8 data in a COBOL variable. For example, if you have UTF-8 data in a PIC X variable, COBOL thinks that the data is EBCDIC and the data could get corrupted. Even something as simple as moving this UTF-8 value from one variable to another variable could corrupt the data, because COBOL pads the variable with X'40' for EBCDIC instead of X'20' for UTF-8.
- Store your data in Db2 using the same encoding scheme as the input data. You gain CPU savings in processing because Db2 and COBOL are both using UTF-16 data or are both using UTF-8 data, and no conversions are needed.
- Use the Db2 coprocessor to prepare your application.
- Specify the appropriate CCSID for your COBOL application source and data according to the instructions in "Specifying a CCSID for your application" on page 54.

  Recommendation: Use the ENCODING bind option to specify the CCSID of the data. This option typically yields the best performance. However, depending on the situation, you might consider the other options for "Specifying a CCSID for your application" on page 54.
- Specify ENCODING UNICODE as a bind option in these situations:
  - Your program uses Db2 variables that are declared as Unicode UTF-16 variables, and specifies the COBOL compiler option SQLCCSID.
  - Your program uses Db2 variables that are declared as Unicode UTF-8 variables.

Related concepts

Processing UTF-8 data using UTF-16 (national) data types (Enterprise COBOL for z/OS Programming Guide)

Processing UTF-8 data by using UTF-8 data types (Enterprise COBOL for z/OS Programming Guide)

Related tasks

Specifying CCSIDs for COBOL applications when using the Db2 coprocessor

If you are using the Db2 coprocessor to prepare a COBOL application with SQL statements, use the COBOL compiler to specify the CCSID of the application source code. For optimal performance, use Db2 to specify the CCSID of the application data in SQL statements.

Related reference

Enterprise COBOL for z/OS

Using national data (Unicode) in COBOL (Enterprise COBOL for z/OS Programming Guide)

Compiler options (COBOL) (Enterprise COBOL for z/OS Programming Guide)

Db2 considerations (Enterprise COBOL for z/OS Programming Guide)

Processing Unicode data in PL/I applications

PL/I and Db2 for z/OSsupport UTF-16 and UTF-8 data.

Procedure

To process Unicode data in PL/I applications for Db2 for z/OS, consider the following recommended actions.

- Use the WIDECHAR data type. This data type supports UTF-16 data in PL/I.

  Although PL/I supports UTF-8 data, you can still use a PL/I application to retrieve UTF-8 data from Db2 into a WIDECHAR variable. For example, if you retrieve data from the Db2 catalog, and the host variable has data type WIDECHAR, Db2 converts the Unicode data for the PL/I application from UTF-8 to UTF-16.
- Use UTF-16 for your Unicode data in your PL/I application and store your application Unicode data in Db2 in UTF-16. This format often requires more space than UTF-8. However, you might gain CPU savings in processing because Db2 and PL/I are both using UTF-16, and no conversions are needed. For additional Db2 CCSID resolution during bind processing and to achieve optimal performance, refer to Character conversion (Introduction to Db2 for z/OS).
- Prepare your application with the Db2 coprocessor.
- Specify the appropriate CCSID for your PL/I application source and data.
- Ensure that your ENCODING bind option matches the data. Depending on the situation, you might consider the other options that are described in “Specifying a CCSID for your application” on page 54.

Related tasks

Specifying CCSIDs for PL/I applications when using the Db2 coprocessor
If you are using the Db2 coprocessor to prepare a PL/I application with SQL statements, use the PL/I compiler to specify the CCSID of the application source code. For optimal performance, use Db2 to specify the CCSID of the application data in SQL statements.

Related reference

Enterprise PL/I for z/OS

Processing Unicode data in C/C++ applications

C/C++ supports UTF-16 data. C/C++ also supports UTF-32 data, but Db2 for z/OS does not.

About this task

Db2 for z/OS, however, supports UTF-8 and UTF-16 data.

Procedure

To process Unicode data in C/C++ applications for Db2 for z/OS:

- For UTF-16 data, use the data type char16_t and prefix these literal values with u.

  In C, char16_t is defined inside the <uchar.h> header. In C++, char16_t is a separate built-in type.
- For SBCS UTF-8 data (UTF-8 data that corresponds to only the first 128 code points in Unicode), specify the ASCII compiler option. When you specify this option, the compiler converts all data to ISO8859-1 (CCSID 819).

  Restriction: You must have an XPLINK application to use the ASCII compiler option.
- If you are using UTF-16 data, store your data in Db2 in UTF-16. This format often requires more space than UTF-8. However, you gain CPU savings in processing, because Db2 and C/C++ are both processing in the UTF-16, and no conversions are needed.
- Specify the appropriate CCSID for your C/C++ application source and data according to the instructions in “Specifying a CCSID for your application” on page 54.

Related reference

The Unicode standard (C/C++) (XL C/C++ Language Reference)
Compiler Options (C/C++) (XL C/C++ User's Guide)

Java applications and Unicode data

Java is Unicode-based, and all character processing inside a Java application occurs in Unicode. Character data that is not already in Unicode must be converted before being passed to a Java application. These conversions are handled by Db2 or by the JDBC driver and are transparent to the application.

You can also pass binary data to a Java application to convert into character data. (This statement assumes that you provide the correct Java encoding.)

From a Java programming perspective, you are manipulating objects and do not need to be concerned with the underlying encoding. However, when your Java application communicates with another technology, such as Db2 for z/OS, conversion might occur. This conversion is handled by Db2 or the JDBC driver, but you should be aware of any conversion costs.

The conversion depends on how you use the driver and how your data is stored in Db2. With IBM Data Server Driver for JDBC and SQLJ type 2 connectivity on Db2 for z/OS, the driver sends the data in the target server encoding scheme. With IBM Data Server Driver for JDBC and SQLJ type 4 connectivity, the driver sends the data in UTF-8.

IBM Data Server Driver for JDBC and SQLJ type 2 connectivity on Db2 for z/OS uses an SQLDA override to tell Db2 if the encoding scheme is different than the one that was specified at bind time. IBM Data Server Driver for JDBC and SQLJ type 4 connectivity uses DRDA data flows to describe the data. Because this environment is a DRDA environment, Db2 does not use the ENCODING bind option to determine the CCSID of the data or to encode data.

Java can handle both big endian and little endian data. (This statement assumes that you provide the correct Java encoding.)

Related concepts

DRDA character type parameters in Unicode
Remote Db2 applications can send and receive DRDA command and reply message parameters that contain character type data encoded in Unicode CCSID 1208 (UTF-8). Using Unicode instead of EBCDIC for these DRDA parameters can improve performance and avoid potential character conversion errors.

Related reference

SQL descriptor area (SQLDA) (Db2 SQL)

Green screen applications and Unicode data

Green screen applications are applications that run on 3270 terminal emulators. These applications do not support Unicode data.

If you migrate your Db2 data to Unicode, consider the following affects on any green screen applications:

Decreased performance
Green screen applications have an EBCDIC encoding scheme. Thus, character conversion might occur between the Db2 data in Unicode and the application. This conversion can increase performance overhead.

Data loss
Unicode data might be lost in the output, unless the content of the data is somehow controlled to ensure that the data is convertible to the EBCDIC CCSID that is used by the 3270 application.

To decide how to handle these problems, consider the reason that you are converting data to Unicode. Is your purpose to accommodate international data or to allow for expansion and flexibility in the future or something else? Knowing your purpose for converting to Unicode can help you choose the appropriate solution for your green screen applications.

If it is acceptable to not have the fields display correctly, you can leave the application as is. For example, some internal reports include names, but they are not required, such as a bank report that lists the top 10 customers by deposit. In this case, the name is a “nice to have” field in the report, but not necessary.

If your application is an output only device, and data is not updated, one possible solution is to use *romanization*. Romanization is the process of creating the Latin representation of a word. To implement this solution, you can have one column for the original data and one column for the phonetic pronunciation in the Latin-1 alphabet. For example, one column might contain Андрей and another column, the romanization column, can contain Andrei. One practical implementation of this solution is in a banking situation. It might be acceptable for a period of time for tellers to have green screen applications that do not display customer names correctly, but provide the phonetic pronunciation. You might need to add logic to be prevent the tellers from updating names, addresses and other information if the teller device is not capable of correctly representing all data.

If you need to display international characters properly, a possible solution is to add a presentation layer to your environment. Consider migrating to a client/server environment, such as the following examples:

- Use CICS Transaction Gateway to access CICS to then access Db2 for z/OS.
- Use an IMS or CICS application that uses WebSphere MQ to access Db2 for z/OS.

Variant characters

Variant characters are characters that correspond to different code points across a given set of code pages. For example, the character # is variant. It corresponds to code point X'7B' in CCSIDs 37, 273, 500, and 1047. However, this character corresponds to code point X'4A' in CCSID 277.

An *invariant character* is a character that corresponds to the same code point regardless of CCSID.

Ideally, you should use invariant characters when possible. However, if you do use variant characters, ensure that Db2 uses the correct CCSID to interpret them.

For example, consider the following national characters: #, @, and $. Although you can use these characters in object identifiers, you should be aware that they are all variant characters. The following table shows the corresponding hexadecimal code point values for these characters in several different code pages.

Table 30. Variant characters that you can use in identifiers

Character	Corresponding hexadecimal value by code page				
	CCSID 37	CCSID 500	CCSID 1047	CCSID 277	CCSID 273
#	X'7B'	X'7B'	X'7B'	X'4A'	X'7B'
@	X'7C'	X'7C'	X'7C'	X'80'	X'B5'
$	X'5B'	X'5B'	X'5B'	X'67'	X'5B'

You need to be careful when you use these characters in identifiers, such as package names, table space names, index space names, and field procedure names. All of these objects have corresponding data sets, DBRMs, or load modules that are defined in z/OS with corresponding names. Problems can occur if you use a different CCSID when the object is created than when it is referenced. In this case, the corresponding data sets, DBRMs, or load modules might not be found in z/OS because of the variant characters in the names.

Another example of a variant character that might cause problems is the double quotation mark ("). In the Turkish code page CCSID 1026 this character corresponds to code point X'FC'. However, this code point is not the same in other EBCDIC code pages.

Also avoid using variant characters in SQL statements. For example, suppose that you want to use an operator to mean "not equal." Coding <> is the best choice, because these characters are invariant across most EBCDIC CCSIDs. However, depending on the situation, Db2 might tolerate other operators for "not equal" such as !=, or ¬=. For details about the conditions that need to be satisfied for Db2 to tolerate

those operators, see Basic predicate (Db2 SQL). Even if these conditions are satisfied, the exclamation point character (!) and the not character (¬) are variant and can therefore cause other problems. For example, these characters might not be displayed correctly on a client. Also, you might have conversion issues if the SQL statement is copied from the catalog or read by another system.

To prevent such problems with variant characters, use the following recommendations.

Best practices:

- Use invariant characters in identifiers and SQL statements.
- When you name Db2 objects, use only those characters that you can type on your keyboard. Do not use hexadecimal values in object names. Doing so can unnecessarily complicate your applications and queries.
- Use CONCAT instead of || when you need to concatenate values.
- Use <> to mean "not equal" instead of != or ¬=.
- Do not use variant hexadecimal code points from another code page. Doing so might cause conversion errors.

Related reference

Code point differences between EBCDIC CCSIDs
Although many EBCDIC code pages are similar, code points for certain characters vary from code page to code page. These characters are called variant characters and can potentially cause problems.

Related information

Invariance of the Syntactic Character Set in Basic SBCS Encoding Structures

DRDA character type parameters in Unicode

Remote Db2 applications can send and receive DRDA command and reply message parameters that contain character type data encoded in Unicode CCSID 1208 (UTF-8). Using Unicode instead of EBCDIC for these DRDA parameters can improve performance and avoid potential character conversion errors.

Prior to DB2 10, remote applications passed DRDA command and reply message parameters that contain character type data in EBCDIC. These applications might incur additional CPU costs and character conversion errors for the following reasons:

- Db2 for z/OS stores metadata and catalog data in Unicode (UTF-8). Therefore, Db2 converts incoming DRDA EBCDIC data to Unicode (UTF-8).
- The IBM Data Server driver or client must convert DRDA character type data to EBCDIC before sending it to Db2 for z/OS. The driver or client must also convert the data that is received from Db2 for z/OS, in EBCDIC, before returning it to the application.
- Other remote applications might need to convert the DRDA parameters to and from EBCDIC.

Passing these character type parameters in Unicode removes this extra conversion step.

From an application programming perspective, you do not need to perform any extra action to send DRDA character type parameters in Unicode. Db2 for z/OS automatically negotiates use of Unicode data with remote client systems that support the exchanging of DRDA character type data parameters in Unicode (UTF-8).

Because of this new ability to pass DRDA character type parameters in Unicode, potential problems might exist with certain package names and collection IDs that contain special characters. To prevent these problems, run the premigration queries.

Related tasks

Run premigration queries (DSNTIJPM) (Db2 Installation and Migration)

Chapter 6. Debugging CCSID and Unicode problems

Some errors are obviously a problem with a CCSID or Unicode object. In other cases, Db2 returns unexpected data and you need to check if a CCSID is the cause of the problem. In these cases, you might not be using Unicode data or doing anything with CCSIDs other than accepting the default values.

Procedure

Consider the symptoms and possible solutions as shown in the following table:

Table 31. Possible solutions to Unicode and CCSID problems

Symptom	Possible solution
A single character is displayed incorrectly.	A CCSID is probably set incorrectly somewhere. Check the following settings: • Ensure that your subsystem CCSIDs are correct. See "Finding the CCSID values of your data sources" on page 27 and "Specifying subsystem CCSIDs" on page 30. If you suspect that you need to change one of your subsystem CCSID values, call IBM Support. • Ensure that the application that you are using is bound with the correct CCSID. Use the ENCODING bind option. For example, if you are using SPUFI, make sure that the SPUFI package is bound with the CCSID that matches the one on your terminal emulator. See "Specifying a CCSID for your application" on page 54. Also try displaying the character in hexadecimal format to see if you can determine what encoding the character is in. Knowing the encoding can also help IBM Support, if you need to contact them.
A conversion that you need is not defined in z/OS Unicode Services. You might have received SQLCODE 332 or message DSNT552I.	Add the missing conversion definition. See "Setting up z/OS Unicode Services for Db2 for z/OS" on page 35.
Lowercase special characters do not become uppercase.	Ensure that you are specifying the correct locale. See "Performing culturally correct case conversions" on page 78.
An insert operation of EBCDIC or ASCII data into a Unicode table fails.	Ensure that the column size is large enough to handle any possible data expansion. See "Potential problems when inserting non-Unicode data into a Unicode table" on page 92.

© Copyright IBM Corp. 2003, 2020

Table 31. Possible solutions to Unicode and CCSID problems (continued)

Symptom	Possible solution
Object names are unreadable in Db2 utility listings	Make sure that the values that are set in DSNHMCID match those values in DSNHDECP. CCSID settings in DSNHMCID are used for certain messages. DSNHMCID settings need to match those same settings in the DSNHDECP load module that the Db2 subsystem is using. For more information about DSNHMCID, see Job DSNTIJUZ and the subsystem parameter load module, application defaults load module, and DSNHMCID (Db2 Installation and Migration).

Potential problems when inserting non-Unicode data into a Unicode table

If you insert EBCDIC or ASCII data into a Unicode table, the data is converted to Unicode. The length of this converted data might increase so much that it causes the operation to fail.

If the source encoding scheme is EBCDIC or ASCII and the target encoding scheme is UTF-8, the worst-case expansion is three times the original. To allow for this worst-case expansion, in the CREATE TABLE statement , declare your UTF-8 columns to be three times the size of your ASCII or EBCDIC columns. You might also want to make your columns varying length so that Db2 does not need to perform padding and truncation on the columns when the length changes due to conversion.

For more information about determining the appropriate column length, see “Estimating the column size for Unicode data” on page 47.

Related concepts

Expanding conversion
An *expanding conversion* occurs when the length of the converted string is greater than that of the source string.

Related tasks

Creating a Unicode table
If you plan to store Unicode data, create Unicode tables. If you try to insert Unicode data into an ASCII or EBCDIC table, data might be lost, unless you use escaped data.

Appendix A. Db2 utilities and Unicode support

You can run Db2 utilities on Unicode data, request that Db2 utilities return data in Unicode, and write utility control statements in Unicode.

More specifically, you can perform the following tasks with Db2 utilities and Unicode data:

- You can load Unicode data into your tables by using the LOAD utility with the UNICODE option. The target table does not need to be a Unicode table. You can load Unicode data into an ASCII or EBCDIC table. Likewise, you can load ASCII or EBCDIC data into a Unicode table. However, in these cases, Db2 converts the input data to the CCSID of the table space before loading it.
- You can use the cross-loader function to load data from a EBCDIC, ASCII, or Unicode source table to an EBCDIC, ASCII, or Unicode target table. If the encoding scheme of the source table is different than the target table, Db2 converts the input data to the encoding scheme of the target table.
- You can unload data in Unicode format by using the UNLOAD utility with the UNICODE option or the IBM Db2 High Performance Unload tool.

  Restriction: With the UNLOAD utility, you cannot:

  – Unload ASCII or EBCDIC SBCS data to UTF-16 output fields.
  – Unload UTF-16 data to ASCII or EBCDIC SBCS output fields.
  – Unload UTF-8 data to UTF-16 output fields.
  – Unload UTF-16 data to UTF-8 output fields.

  For these situations, use the High Performance Unload tool.
- You can write utility control statements in either EBCDIC or UTF-8.
- You can use the Db2-supplied stored procedure DSNUTILU to invoke a Db2 utility from an application program with a utility control statement that is written in Unicode. Alternatively, you can use the DSNUTILS stored procedure with an EBCDIC utility control statement.

Related concepts
Utility control statements (Db2 Utilities)
Related tasks
Loading data by using the cross-loader function (Db2 Utilities)
Related reference
LOAD (Db2 Utilities)
UNLOAD (Db2 Utilities)
Db2 High Performance Unload for z/OS
DSNUTILU stored procedure (Db2 SQL)
DSNUTILS stored procedure (deprecated) (Db2 SQL)

© Copyright IBM Corp. 2003, 2020

Appendix B. EXPLAIN Unicode support

You can use Db2 EXPLAIN to capture access path information for your queries. This information is stored in the Db2 EXPLAIN tables, which are encoded in UTF-8.

PSPI

When you retrieve data from the EXPLAIN tables, be aware that the data is encoded in UTF-8.

The following EXPLAIN table columns store encoding and CCSID information:

PLAN_TABLE columns:

TABLE_ENCODE
Indicates the encoding scheme of the statement. If the statement represents a single CCSID set, the column contains 'E' for EBCDIC, 'A' for ASCII, or 'U' for Unicode. If the statement is a multiple CCSID set statement, the column is set to 'M' for multiple CCSID sets.

TABLE_SCCSID
Contains the SBCS CCSID value of the table or zero if the TABLE_ENCODE column is 'M.'

TABLE_MCCSID
Contains the Mixed CCSID value of the table or zero if the TABLE_ENCODE column is 'M.'

TABLE_DCCSID
Contains the DBCS CCSID value of the table or zero if the TABLE_ENCODE column is 'M.'

DSN_STATEMNT_TABLE column:

STMT_ENCODE
Indicates the encoding scheme of the statement. If the statement represents a single CCSID set, the column contains 'E' for EBCDIC, 'A' for ASCII, or 'U' for Unicode. If the statement is a multiple CCSID set statement, the column is set to 'M' for multiple CCSID sets.

PSPI

Related concepts

Investigating SQL performance by using EXPLAIN (Db2 Performance)

© Copyright IBM Corp. 2003, 2020

Appendix C. Db2 ODBC Unicode support

Your Db2 for z/OS ODBC programs can manipulate Unicode data and report the CCSID settings of the subsystem.

If your application manipulates UTF-8 data, set the initialization keyword CURRENTAPPENSCH to UNICODE or any Unicode CCSID value. When you set CURRENTAPPENSCH for Unicode data, you can use the following items for UTF-8 data:

- The generic APIs, such as SQLColumnPrivileges. When CURRENTAPPENSCH is set to UNICODE, these APIs accept UTF-8 string arguments and return all character string data in the result set in UTF-8.
- C data type SQL_C_CHAR. When CURRENTAPPENSCH is set to UNICODE, the Db2 for z/OS ODBC driver assumes UTF-8 data for SQL_C_CHAR. This data type is used by the APIs SQLBindCol(), SQLBindParameter(), and SQLGetData().

If your application manipulates UTF-16 data, use APIs with the suffix W, which are called wide APIs, on that data. Any generic API that accepts character string arguments has a wide API counterpart. For example, the corresponding wide API for the SQLConnect() API is SQLConnectW(). Wide APIs accept UTF-16 string arguments only. The initialization keyword CURRENTAPPENSCH does not affect these wide APIs. Regardless of what you specify for CURRENTAPPENSCH, these wide APIs always expect Unicode UTF-16 data. You can also use the SQL_C_WCHAR data type for UTF-16 data. Like wide APIs, SQL_C_WCHAR also assumes UTF-16 data, regardless of what you specify for CURRENTAPPENSCH.

You can use the SQLGetInfo() API with certain attributes, such as SQL_ASCII_SCCSID, to query the CCSID settings of the Db2 subsystem.

Related concepts

Application encoding schemes and Db2 ODBC (Db2 Programming for ODBC)

Related reference

Db2 ODBC initialization keywords (Db2 Programming for ODBC)

C and SQL data types (Db2 Programming for ODBC)

SQLGetInfo() - Get general information (Db2 Programming for ODBC)

© Copyright IBM Corp. 2003, 2020

Appendix D. IBM Db2 Tools for z/OS Unicode support

You can use IBM Db2 Tools for z/OS on Unicode data, objects, and applications.

Db2 High Performance Unload for z/OS
You can use this tool to unload table data in any encoding scheme. You can also change the encoding scheme for its output.

IBM Data Studio
You can use this tool to create and modify applications that handle UTF-8 and UTF-16 data. You can also use this tool to create, manage, and access Unicode objects, such as databases, table spaces, and tables, in Db2 for z/OS.

Db2 Query Management Facility (QMF)
You can use this tool to query and report on Unicode data in Db2 for z/OS. Use QMF for Workstation for complete Unicode support.

Db2 Table Editor for z/OS
You can use the Db2 Table Editor for z/OS to create, view, or update Unicode data in Db2 for z/OS.

Related reference

Db2 High Performance Unload for z/OS

IBM Data Studio product documentation (IBM Data Studio, IBM Optim Database Administrator, IBM infoSphere Data Architect)

Db2 Query Management Facility (QMF) information

Db2 Table Editor for z/OS

Db2 Tools for z/OS

Related information

Db2 Tools for z/OS Product Documentation

© Copyright IBM Corp. 2003, 2020

Appendix E. The International Components for Unicode

The International Components for Unicode (ICU) is a set of C/C++ and Java libraries for Unicode support and software internationalization. ICU is an open source project that is sponsored by IBM and provides Unicode services on many platforms.

For Db2 for z/OS, ICU is provided as part of the Accessories Suite. It is called by certain features of Db2 for z/OS and the Accessories Suite, such as Spatial Support, that require these Unicode and internationalization functions.

Related information

International Components for Unicode (ICU)

© Copyright IBM Corp. 2003, 2020

Appendix F. SYSSTRINGS catalog table

The SYSSTRINGS table contains information about character conversion. Each row describes a conversion from one coded character set to another. The schema is SYSIBM.

Also refer to z/OS C/C++ Programming Guide for information on the additional conversions that are supported.

Each row in the table must have a unique combination of values for its INCCSID, OUTCCSID, and IBMREQD columns. Rows for which the value of IBMREQD is N can be deleted, inserted, and updated subject to this uniqueness constraint and to the constraints imposed by a VALIDPROC defined on the table. An inserted row could have values for the INCCSID and OUTCCSID columns that match those of a row for which the value of IBMREQD is Y. Db2 then uses the information in the inserted row instead of the information in the IBM-supplied row. Rows for which the value of IBMREQD is Y cannot be deleted, inserted, or updated. For information about the use of inserted rows for character conversion, see *Db2 Installation Guide*.

Db2 has two methods for character conversions and applies them in the following order:

1. Conversions specified by the various combinations of the INCCSID and OUTCCSID columns in the SYSIBM.SYSSTRINGS catalog table.
2. Conversions provided by z/OS support for Unicode. For more information, see z/OS Support for Unicode: Using Conversion Services.

If neither of these methods can be used for a particular character conversion, Db2 returns an error.

Column name	Data type	Description	Use
INCCSID	INTEGER NOT NULL	The source CCSID for the character conversion represented by this row. The value of the source CCSID must be in the range of 1 to 65533 and must not be the same as the value for the OUTCCSID column.	G
OUTCCSID	INTEGER NOT NULL	The target CCSID for the character conversion represented by this row. The value of the target CCSID must be in the range of 1 to 65533 and must not be the same as the value for the INCCSID column.	G

© Copyright IBM Corp. 2003, 2020

Column name	Data type	Description	Use
TRANSTYPE	CHAR(2) NOT NULL	Indicates the nature of the conversion. Values can be: **GG** GRAPHIC to GRAPHIC **MM** EBCDIC MIXED to EBCDIC MIXED **MS** EBCDIC MIXED to SBCS **PM** ASCII MIXED to EBCDIC MIXED **PS** ASCII MIXED to SBCS **SM** SBCS to EBCDIC MIXED **SS** SBCS to SBCS **MP** EBCDIC MIXED to ASCII MIXED **PP** ASCII MIXED to ASCII MIXED **SP** SBCS to ASCII MIXED	G
ERRORBYTE	CHAR(1) FOR BIT DATA (Nulls are allowed)	The byte used in the conversion table as an error byte. Any non-null value that is specified for the ERRORBYTE column must not be the same as the value that is specified for the SUBBYTE column. Null indicates the absence of an error byte.	S
SUBBYTE	CHAR(1) FOR BIT DATA (Nulls are allowed)	The byte used in the conversion table as a substitution character. Any non-null value that is specified for the SUBBYTE column must not be the same as the value that is specified for the ERRORBYTE column. Null indicates the absence of a substitution character.	S
TRANSPROC	VARCHAR(24) NOT NULL WITH DEFAULT	The name of a module or blanks. A nonblank value must conform to the rules for z/OS program names. If IBMREQD is 'N', a nonblank value is the name of a conversion procedure provided by the user. The first five characters of the name of a user-provided conversion procedure must not be 'DSNXV'; these characters are used to distinguish user-provided conversion procedures from Db2 modules that contain DBCS conversion tables. If IBMREQD is 'Y', a nonblank value is the name of a Db2 module that contains DBCS conversion tables.	G

Column name	Data type	Description	Use
IBMREQD	CHAR(1) NOT NULL	A value of Y indicates that the row came from the basic machine-readable material (MRM) tape. **Value** **Meaning** **B** Version 1R3 dependency indicator, not from the machine-readable material (MRM) tape **C** Version 2R1 dependency indicator, not from MRM tape **D** Version 2R2 dependency indicator, not from MRM tape **E** Version 2R3 dependency indicator, not from MRM tape **F** Version 3R1 dependency indicator, not from MRM tape **G** Version 4 dependency indicator, not from MRM tape **H** Version 5 dependency indicator, not from MRM tape **I** Version 6 dependency indicator, not from MRM tape **J** Version 6 dependency indicator, not from MRM tape **K** DB2 Version 7 dependency indicator, not from MRM tape **L** DB2 Version 8 dependency indicator, not from MRM tape **M** DB2 9 dependency indicator, not from MRM tape **O** DB2 10 dependency indicator, not from MRM tape **P** Db2 11 dependency indicator, not from MRM tape **N** Not from MRM tape, no dependency The value in this field is not a reliable indicator of release dependencies.	G
TRANSTAB	VARCHAR(256) FOR BIT DATA NOT NULL WITH DEFAULT	Either a 256-byte conversion table or an empty (0 length) string.	S

Information resources for Db2 11 for z/OS and related products

Information about Db2 11 for z/OS and products that you might use in conjunction with Db2 11 is available online in .

You can find the complete set of product documentation for Db2 11 for z/OS in IBM Knowledge Center.

You can also download other PDF format manuals for Db2 11 for z/OS from in PDF format manuals for Db2 11 for z/OS (Db2 Home).

© Copyright IBM Corp. 2003, 2020

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing Legal and Intellectual Property Law IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785 US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

© Copyright IBM Corp. 2003, 2020

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as shown below:

© (*your company name*) (*year*).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. (*enter the year or years*).

Programming interface information

This information is intended to help you handle international data when working in a Db2 11 for z/OS environment. This information also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by Db2 11 for z/OS.

General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of Db2 11 for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

GUPI General-use Programming Interface and Associated Guidance Information... GUPI

Product-sensitive Programming Interface and Associated Guidance Information

Product-sensitive Programming Interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive Programming Interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:

PSPI Product-sensitive Programming Interface and Associated Guidance Information... PSPI

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered marks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at: http://www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions:

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Glossary

The glossary is available in IBM Knowledge Center.

See the Glossary topic for definitions of Db2 for z/OS terms.

© Copyright IBM Corp. 2003, 2020

Index

Numerics

3270 applications
  Unicode data 88

A

access paths
  for Unicode data 52
accessibility
  keyboard viii
  shortcut keys viii
AGCCSID 31
AMCCSID 31
APPENSCH
  description 31
  when Db2 uses 54
application data
  determining CCSID 27
application encoding scheme
  definition 22, 54
application programming
  querying the catalog 22
  recommendations 53
  Unicode 53
applications
  examples of specifying CCSIDs 59
  specifying a CCSID 54
ASCCSID 31
ASCII
  description 7
ASCII table
  inserting Unicode data 49
ASCII_STR function
  generating escaped data 82
authorization processes
  Unicode data 21

B

best practices
  for coding queries 89
big endian
  description 14
BIT data
  Unicode tables 44
byte order formats
  big endian 14
  little endian 14

C

C/C++
  LOCALE compiler option 66
  processing Unicode data 87
  specifying CCSID 66
C/C++ *(continued)*
  SQL compiler option with CCSID suboption 66
C/C++ source code
  determining CCSID 27
canonically equivalent characters
  description 84
case conversions
  specifying culturally correct rules 78
catalog tables
  CCSID information 67
  encoding scheme 22
  in Unicode 21
  querying 22
  SYSSTRINGS
    contents 103
CCSID 1047
  compared to 500 9
  compared to CCSID 37 9
CCSID 1140
  code points 4
  compared to CCSID 37 60
CCSID 1200
  description 13
  for table column 44
CCSID 1208
  description 13
  for table column 44
CCSID 367
  code points 11
  description 13
  for table column 44
  relationship to Unicode 10
CCSID 37
  code points 4
  compared to CCSID 1047 9
  compared to CCSID 1140 60
  compared to CCSID 500 9
CCSID 500
  compared to CCSID 1047 9
  compared to CCSID 37 9
CCSID set
  specifying Db2 defaults 30
CCSID SQL processing option
  description 56
  setting 54
CCSID UNICODE clause
  CREATE statements 44
CCSID0 63
CCSIDs
  C/C++ 66
  COBOL applications 60
  conversions 15
  description 4
  determining for data source 27
  determining for Db2 data 67
  determining subsystem values 33
  determining value for string 68

CCSIDs *(continued)*
examples of specifying in applications 59
for PL/I applications 63
in EXPLAIN tables 95
multiple in SQL statement 68
specifying for an application 54
specifying for objects 34
specifying for subsystem 30
specifying in Db2 29
subsystem defaults 31
Unicode data 13
where to define valid conversions 25
character columns
Unicode tables 44
character conversion
contracting conversion 18
data loss 16
defining 40
definition 1
determining length 16
effects 16
enforced subset 19
ensuring accurate conversions 25
expanding conversion 16
in Db2 25
Java applications 88
occurrences 15
performance 16
round-trip conversion 18
substitution character 19
SYSIBM.SYSSTRINGS catalog table 103
terminology 1
types 16
where to define valid conversions 25
character conversion definitions
adding 40
checking 40
Chinese 38
finding 40
Japanese 38
Korean 38
SYSIBM.SYSSTRINGS 26
z/OS Unicode Services 36, 38
character data representation architecture (CDRA)
description 4
character repertoire
definition 11
Chinese character sets
conversion definitions 38
CICS Transaction Gateway
determining CCSID 27
COBOL
CODEPAGE compiler option 60
NOSQLCCSID 60
processing Unicode data 86
specifying CCSIDs 60
SQL compiler option with CCSID suboption 60
SQLCCSID 60
COBOL source code
determining CCSID 27
code pages
description 4
code points
description 4
code points *(continued)*
differences in EBCDIC CCSIDs 9
code units
size for UTFs 11
CODEPAGE compiler option
COBOL 60
PL/I 63
CODEUNITS16
description 73
CODEUNITS32
description 73
collation names 76
COLLATION_KEY function
creating index with 76
specifying sorting sequence 76
columns
estimating length for Unicode data 47
compatibly equivalent characters
description 84
compilers
specifying CCSID 54
contracting conversion
description 18
conversion image
creating 35
description 35
conversion tables
definition 35
converting data
to Unicode 49
cross-loader function
Unicode tables 93
CURRENT APPLICATION ENCODING SCHEME special register
description 56
setting 54
CURRENT LOCALE LC_CTYPE special register
setting default locale 80
specifying casing rules 78
CURRENTAPPENSCH initialization keyword
specifying ODBC application encoding scheme 97

D

data sources
determining CCSID 27
databases
specifying CCSID 34
Db2 coprocessor
setting the application CCSID 54
Db2 objects
encoding scheme 34
specifying CCSIDs 34
Db2 precompiler
setting the application CCSID 54
Unicode parsing 21
Db2 processes
in Unicode 21
Db2 Query Management Facility (QMF)
Unicode support 99
Db2data
determining CCSID 67
DBRM
in Unicode 21

debugging problems
with CCSIDs 91
with Unicode data 91
DECLARE VARIABLE statement
description 56
setting host variable CCSID 54
decoding
definition 11
DESCRIBE statement
checking CCSID of string 68
disability viii
DISPLAY UNI command
checking character conversion definitions 40
checking supported Unicode characters 13
DRDA
effect of ENCODING bind option 54
setting CCSID 54
DRDA Unicode parameters 90
DSNHDECP
CCSID information 30
subsystem CCSIDs 31
DSNJU004
determining subsystem CCSID values 33
DSNTIJUZ job
specifying CCSIDs 30
DSNTIPF installation panel
specifying CCSIDs 30
DSNUTILU
invoking utilities 93

E

EBCDIC
CCSID code point differences 9
description 8
EBCDIC sorting sequence
differences from Unicode 71
EBCDIC table
inserting Unicode data 49
EBCDIC_STR function
generating escaped data 82
encoding
definition 11
ENCODING bind option
C/C++ 66
COBOL 60
description 56
DRDA 54
PL/I 63
setting 54
encoding schemes
ASCII 7
description 6
EBCDIC 8
subsystem defaults 31
Unicode 10
endianness
description 14
enforced subset conversion
description 19
ENSCHEME 31
ERRORBYTE column of SYSSTRINGS catalog table 103
escaped data
description 82
escaped data *(continued)*
generating 82
expanding conversion
description 16
effects 16
when loading Unicode tables 92
EXPLAIN
Unicode support 95
EXPLAIN tables
CCSID information 95
in UTF-8 95

F

fixed-length strings
expanding conversions 16
fixed-length variables
expanding conversions 16
FTP data
determining CCSID 27
functions
calculating length 73
specifying CCSID 34

G

GCCSID 31
general-use programming information, described 110
GETVARIABLE
determining subsystem CCSID values 33
graphic columns
Unicode tables 44
green screen applications
Unicode data 88
GUPI symbols 110

H

High Performance Unload
Unicode support 99

I

IBM Data Studio
Unicode support 99
IBMREQD
SYSIBM.SYSSTRINGS column 26
IBMREQD column
SYSSTRINGS catalog table 103
ICU 101
IFCID
Unicode output 22
image generator
description 35
IMS
determining CCSID 27
INCCSID column of SYSSTRINGS catalog table 103
installation job DSNTIJUZ
specifying CCSIDs 30
installation panel DSNTIPF
specifying CCSIDs 30
International Components for Unicode 101
invariant characters 89

ISPF
determining CCSID 27

J

Japanese character sets
conversion definitions 38
Java
character conversion 88

K

Korean character sets
conversion definitions 38

L

language casing rules
specifying 78
language-specific sorting sequences 76
LE locales 80
length
estimating for Unicode columns 47
expansion when inserting Unicode data 92
specifying how Db2 calculates 73
length functions
effect of contracting conversions 18
effect of expanding conversions 16
links
non-IBM Web sites 111
little endian
description 14
LOAD utility
Unicode data 93
local system
character conversions 15
LOCALE
C/C++ compiler option 66
locales
Db2 functions 78
description 80
LE locales 80
z/OS Unicode Services 80
LOWER function
ensuring culturally correct results 78

M

MCCSID 31
mixed data
Unicode tables 44
multiple CCSIDs
referenced in the same SQL statement 68

N

NOCCSID0 63
Normalization Form Canonical Composition (NFC) 84
Normalization Form Canonical Decomposition (NFD) 84
Normalization Form Compatibly Composition (NFKC) 84
Normalization Form Compatibly Decomposition (NFKD) 84
NORMALIZE_STRING function
ensuring well-formed data 85
NORMALIZE_STRING function *(continued)*
normalizing Unicode strings 84
normalizing
Unicode strings 84
NOSQLCCSID 60

O

object names
contracting conversions 18
expanding conversions 16
objects
determining CCSID 67
OCTETS
description 73
ODBC
Unicode support 97
ORDER BY clause
differences by encoding scheme 71
OUTCCSID column of SYSSTRINGS catalog table 103

P

package names
Unicode 21
Personal Communications
determining CCSID 27
PL/I
CCSID0 63
CODEPAGE compiler option 63
NOCCSID0 63
PP compiler option 66
processing Unicode data 87
specifying CCSIDs 63
SQL Preprocessor 66
PL/I source code
determining CCSID 27
PP compiler option 66
predefined conversion definitions
checking 40
preparing Db2
for character conversion 25
procedures
specifying CCSID 34
product-sensitive programming information, described 110
programming interface information, described 110
PSPI symbols 110

Q

QMF
determining CCSID 27
queries
best practices for coding 89
Queue Managers in WebSphere MQ
determining CCSID 27

R

remote applications
passing DRDA parameters in Unicode 90
remote system
character conversions 15

round-trip conversion
description 18

S

SBCS data
Unicode tables 44
SCCSID 31
setting up Db2
for character conversion 25
SETUNI command
activating conversion image 35
shortcut keys
keyboard viii
sorting data
with language specific rules 76
sorting sequence
EBCDIC 71
language-specific 76
specifying 76
Unicode 71
special characters
typing 48
special registers
Unicode 21
SQL compiler option with CCSID suboption
C/C++ 66
COBOL 60
SQL preprocessor for PL/I 66
SQL statements
different CCSIDs 68
SQL_C_WCHAR data type
ODBC applications 97
SQLCCSID 60
SQLDA
checking CCSID information 68
SQLGetInfo API
retrieving subsystem CCSID information in ODBC applications 97
storage
Unicode, tips 46
string length
determining 73
strings
determining CCSID 68
SUBBYTE column of SYSSTRINGS catalog table 103
substitution character
in character conversions 19
subsystem
CCSIDs 31
determining CCSID values 33
encoding schemes 31
specifying CCSIDs 30
subsystem CCSIDs
retrieving in ODBC applications 97
subsystem default application encoding scheme 31
subsystem default ASCII CCSID 31
subsystem default EBCDIC CCSID 31
subsystem default encoding scheme 31
subsystem default Unicode CCSID 31
supplementary characters
description 85
how Db2 handles 85
syntax diagram
how to read ix
syntax diagram *(continued)*
SYSIBM.SYSSTRINGS
adding character conversion definitions 40
description 26
how Db2 uses for character conversion 25
querying 40
system default application encoding scheme
when Db2 uses 54

T

Table Editor
Unicode support 99
table spaces
specifying CCSID 34
tables
creating in Unicode 44
estimating column size for Unicode data 47
inserting Unicode data 48
specifying CCSID 34
terminology
for character conversion 1
tools
Unicode support 99
trace
Unicode output 22
traces
Unicode 21
TRANSLATE function
ensuring culturally correct results 78
TRANSPROC column of SYSSTRINGS catalog table 103
TRANSTAB column of SYSSTRINGS catalog table 103
TRANSTYPE column of SYSSTRINGS catalog table 103

U

UGCCSID 31
UMCCSID 31
Unicode
application programming 53
CCSIDs 13
converting Db2 data to 49
debugging 91
description 10
differences from EBCDIC 71
EXPLAIN support 95
generating escaped data 82
in Db2 21
normalizing strings 84
ODBC support 97
preserving data 82
problems inserting data 92
sorting sequence 71
storage tips 46
supplementary characters 85
utilities 93
Unicode data
access paths 52
C/C++ applications 87
COBOL applications 86
green screen applications 88
inserting into non-Unicode table 49

Unicode data *(continued)*
PL/I applications 87
storing 43
Unicode on Demand 35
Unicode support
Db2 High Performance Unload for z/OS 99
Db2 Table Editor for z/OS 99
IBM Data Studio 99
IBM tools 99
Unicode tables
creating 44
determining column type 44
estimating column size 47
inserting data 48
Unicode Transformation Formats (UTFs)
Db2 support 11
UNICODE_STR function
interpreting escaped data 82
UNLOAD utility
Unicode data 93
UPPER function
ensuring culturally correct results 78
USCCSID 31
UTF-16
compared to UTF-8 43
Db2 support 43
description 11
endianness 14
UTF-32
description 11
endianness 14
UTF-8
compared to UTF-16 43
Db2 support 43
description 11
endianness 14
UTFdetermining which to use 43
utilities
control statements in Unicode 21
Unicode support 93

V

variant characters 89

W

wide APIs
ODBC UTF-16 data 97

Z

z/OS
determining CCSID 27
z/OS Unicode Services
adding character conversion definitions 40
basic character conversions 36
Chinese, Japanese, and Korean character conversions 38
conversion image 35
how Db2 uses for character conversion 25
locales 80
setting up 35



Product Number: 5615-DB2
5697-P43

SC19-4057-06

