IBM

# DB2 UDB for z/OS Internationalization Guide

# DB2 UDB for z/OS Internationalization Guide

**Note**

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 35.

**First Edition (June 2004)**

This edition applies to Version 8 of IBM DB2 Universal Database Server for z/OS (DB2 UDB for z/OS), product number 5625-DB2, 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.

# Contents

# About this document

This topic contains information that describes this document.

## Who should read this document

This document is primarily intended for those people who are responsible for using character conversion in a DB2 UDB for z/OS environment.

This book assumes that you are familiar with:

- The basic concepts and facilities of DB2 in the z/OS environment
- The basic concepts of Structured Query Language (SQL)

## Product terminology and citations

In this document, DB2 Universal Database™ for z/OS® is referred to as "DB2 UDB for z/OS." In cases where the context makes the meaning clear, DB2 UDB for z/OS is referred to as "DB2®." When this book refers to other books in this library, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM® DB2 Universal Database for z/OS SQL Reference*.)

When referring to a DB2 product other than DB2 UDB for z/OS, this book uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

**DB2** Represents either the DB2 licensed program or a particular DB2 UDB for z/OS subsystem.

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

# Chapter 1. Introduction to character conversion

A *string* is a sequence of bytes that can represent characters. Within a string, all the characters are represented by a common encoding representation. In some cases, it might be necessary to convert these characters to a different encoding representation. The process of conversion is known as *character conversion*. Character conversion, when required, is automatic, and when successful, it is transparent to the application.

## Character conversion terminology

The following list defines some of the terms used for character conversion.

**code point**
A unique bit pattern that represents a character.

**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.

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

**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 an attribute of strings, just as length is an attribute of strings. All values of the same string column have the same CCSID. A CCSID may contain one or more code pages.

**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, and 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.

**ASCII** Acronym for American Standard Code for Information Interchange, an encoding scheme used to represent characters. The term ASCII is used throughout this document to refer to IBM-PC data or ISO 8-bit data.

**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 map characters onto code points, each consisting of 8 bits.

**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 originally established as a fixed-width, 16-bit encoding, to provide enough code points for all the scripts and technical symbols in common usage around the world, plus some ancient scripts.

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

## What is character conversion?

Computers store only numbers. They store letters and other characters by assigning a number to them. A representation of a character in bytes is called a *code point.* A *code page* is a set of code points for a particular character set.

Various encoding schemes, such as ASCII and EBCDIC, were invented to standardize the encoding of a character set. These encoding schemes have several limitations, including:
- No single encoding was adequate for all letters and symbols available. For example, most encoding schemes have one code page to map Japanese characters and another code page to map German characters.
- Encoding schemes often encode data in different positions. For example, the letter A is encoded as X'C1' in EBCDIC, but X'41' in ASCII.

Character data may be represented by different encoding schemes. A number of code pages are defined using the rules of each encoding scheme. In DB2, each code page is identified by a unique coded character set identifier (CCSID). For example, the ASCII encoding scheme is represented by many CCSIDs. All character data has a CCSID.

The CCSID is a number that uniquely identifies one or more pairs of character sets and code pages. The coded character set defines how bit configurations are mapped for character data. A complete description of all IBM-registered CCSIDs and conversion tables exists in *Character Data Representation Architecture Reference and Registry*. For general information about character conversion, see *Character Data Representation Architecture Overview*.

A list of many code pages that IBM supports is available at www.ibm.com/servers/eserver/iseries/software/globalization/codepages.html

*Character conversion* occurs when data is encoded in one encoding scheme and is later converted to another encoding scheme.

## When does character conversion occur?

In client/server environments, character conversion can occur when an SQL statement is executed remotely. Consider, for example, these two cases:
- The values of data that the requester sent to the current server
- The values of data that the current server sent to the requester

In either case, the data could have a different representation at the sending and receiving systems. If the data has a different representation at the systems, conversion may be required.

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

- An overriding CCSID is specified in a DECLARE VARIABLE statement.
- An overriding CCSID is specified in the SQLDA.
- An SQL statement refers to data that is defined with different CCSIDs.
- A mixed-character string is assigned to an SBCS column or host variable. For example, if you have set MIXED=NO in DSNHDECP and you use SPUFI to insert data into a Unicode UTF-8 table (which is MIXED=YES by default), a conversion will occur.
- The value of the ENCODING bind option (static SQL statements) or the CURRENT APPLICATION ENCODING SCHEME special register (for dynamic SQL) is different than the encoding scheme of the data that is being retrieved.
- An ASCII application provides SQL statement text to DB2 in a PREPARE statement. In DB2 Version 7, DB2 converts the statement text to EBCDIC for parsing. In DB2 Version 8, DB2 converts the statement text to Unicode for parsing.

Character conversion is described in terms of CCSIDs of the source and target. With DB2 UDB for z/OS, two methods are used to identify valid source and target combinations and to perform the conversion from one coded character set to another:

- DB2 catalog table SYSIBM.SYSSTRINGS

  Each row in the catalog table describes a conversion from one coded character set to another.
- z/OS Support for Unicode

  For more information about the conversion services that are provided, including a complete list of the IBM-supplied conversion tables, see *z/OS Support for Unicode: Using Conversion Services*.
- Language Environment (DB2 Version 7 only)

## Types of character conversion

Character conversion between two CCSIDs can have an impact on your subsystem. Be aware of the following types of character conversions that might occur:

- Expanding conversion
- Contracting conversion
- Enforced subset conversion
- Round-trip conversion

Expanding conversions and contracting conversions might affect the length of the converted string. Enforced subset conversions and round-trip conversions might affect the contents of the string.

### 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 EBCDIC mixed data. Because of the addition of shift-out and shift-in control characters, an error occurs when an expanding conversion is performed on a fixed-length input host variable that requires conversion from ASCII mixed data to EBCDIC mixed data. Expanding conversions also can occur when string data is converted to or from Unicode.

The solution is to use a varying-length string variable with a maximum length that is sufficient to contain the expansion.

**Example:** In CCSID 819, the character Å is represented by the code point X'C5'. In CCSID 1208, this character is represented by X'C385'.

## 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 data due to the removal of shift characters. Contracting conversions also can occur when string data is converted to or from Unicode.

**Example:** In CCSID 1200, the character Å is represented by the code point X'00C5'. In CCSID 819, this character is represented by X'C5'.

## Enforced subset conversion

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

**Example:** In CCSID 1252, the trademark symbol ™ is represented by the code point X'99'. During an enforced subset character conversion to CCSID 37, this code point is converted to the substitution character X'3F'. During character conversion to CCSID 1252, the character remains a substitution character, represented by the code point X'1A'

## Round-trip conversion

A *round-trip conversion* is designed to preserve characters that are exist in the source CCSID but do not exist in the target CCSID. Round-trip conversions between two CCSIDs assure that all characters that make the ″round trip″ arrive as they were originally, even if the receiving CCSID does not support a given character.

**Example:** In CCSID 1252, the trademark symbol ™ is code point X'99'. During character conversion to CCSID 37, this code point is converted to a control character. Using a round-trip conversion back to CCSID 1252, the trademark symbol is preserved as code point X'99'.

## Introduction to Unicode

In this era of globalization, the ability for systems to be able to handle data from around the world is paramount. However, workstations and servers might use different code pages, depending on the country in which they reside. In effect, they are speaking different languages, making communication difficult.

The design objective of Unicode is to avoid these issues by having a single standard that has a code point mapping for every character in the world. The Unicode Consortium has devised a number of Unicode Transformation Formats (UTFs) to meet this objective. Each UTF is designed for different processing objectives. All UTFs map the full Unicode character repertoire. The different UTFs are as follows:

**UTF-8** UTF-8 is designed to be an Internet encoding transformation format. Each character is encoded as 1 to 4 bytes; the first 128 code points from X'00' to X'7F' are the same as the first 128 code points in ASCII. DB2 uses UTF-8 in the following ways:

- DB2 encodes data in CHAR, VARCHAR, and CLOB columns in UTF-8.
- DB2 parses and precompiles data in UTF-8.
- In DB2 Version 8, the catalog is encoded in UTF-8 after you have converted to new-function mode.

to encode data in CHAR, VARCHAR, and CLOB columns.

**UTF-16**

UTF-16 is a transformation format that is based on 16-bit code units. Most Chinese, Japanese, and Korean characters are encoded with a 2-byte code unit in UTF-16. These characters can be stored in UTF-8, but they use 3 to 4 bytes and thus require additional storage. Some characters that are encoded with a 1-byte code unit in UTF-8, such as the characters A-Z, a-z, 0-9, and some special characters, are encoded with a 2-byte code unit in UTF-16. DB2 uses UTF-16 to encode data in GRAPHIC, VARGRAPHIC, and DBCLOB columns.

**UTF-32**

UTF-32 encodes each character as 4 bytes. DB2 does not use UTF-32.

DB2 supports two implementations of the Unicode encoding standard: UTF-8 and UTF-16. UTF-8 is the installation default; the Unicode CCSID field of panel DSNTIPF is pre-filled with 1208, which is the CCSID for UTF-8. However, during processing, DB2 chooses the appropriate CCSIDs for double-byte and single-byte values. For single-byte character set (SBCS) data, DB2 chooses CCSID 367, which corresponds to the first 128 code points for UTF-8. For double-byte characters set (DBCS) data, DB2 chooses CCSID 1200, which corresponds to UTF-16.

# Chapter 2. Character conversion and CCSIDs in DB2 UDB for z/OS

This topic discusses character conversion and how CCSIDs are used in DB2 UDB for z/OS.

## Character conversion and CCSIDs

To support character conversion, the IBM Distributed Relational Database Architecture (DRDA) uses CCSIDs to label the various character representation schemes.

The CCSID is a number that uniquely identifies one or more pairs of character sets and code pages. The coded character set defines how bit configurations are mapped for character data. A complete description of all IBM-registered CCSIDs and conversion tables exists in *Character Data Representation Architecture Reference and Registry*. For general information about character conversion, see *Character Data Representation Architecture Overview*.

In DB2, all character data is associated with a CCSID. Character conversion is described in terms of CCSIDs of the source and of the target. When you install DB2, you must specify a CCSID for ASCII, EBCDIC, and Unicode DB2 character data. In general, DB2 assumes that data is encoded using the system EBCDIC CCSID unless you specify otherwise.

The CCSIDs of character strings at your site are determined by the CCSIDs that you specify on installation panel DSNTIPF. If the EBCDIC CCSID is not correct, character conversion might produce incorrect results. The correct CCSID is the number that identifies the coded character set that is supported by the I/O devices, 3270 terminal emulators, local applications such as IMS and DB2 QMF, and remote applications such as QMF for Windows at your site. For more information about selecting CCSIDs, see the description of installation panel DSNTIPF in *DB2 Installation Guide*.

DB2 performs most character conversion automatically, based on CCSIDs. If character conversion must occur, first DB2 searches the catalog table SYSIBM.SYSSTRINGS for a conversion. If SYSIBM.SYSSTRINGS does not define a conversion, DB2 uses z/OS Unicode Conversion Services. If neither DB2 nor z/OS Unicode Conversion Services provides a conversion for a certain combination of source and target CCSIDs, you receive an error message. To correct the problem, you need to understand the rules for assigning source and target CCSIDs in SQL operations. See Appendix A, “Relationship between a string and its encoding scheme,” on page 19 for more information about these rules.

## Choosing CCSIDs

If you specify MIXED DATA = NO on panel DSNTIPF, you can use any compatible SBCS CCSID in the EBCDIC CODED CHAR SET and ASCII CODED CHAR SET fields. If you specify MIXED DATA = YES on panel DSNTIPF, you must use the appropriate mixed CCSID. By specifying a CCSID for mixed data, you also receive system CCSIDs for SBCS and DBCS (graphic) data. More information about selecting the appropriate mixed CCSID is available in Appendix A of *DB2 Installation Guide*.

See the description of installation panel DSNTIPF and Appendix A in *DB2 Installation Guide* for more information about mixed data types.

## Setting up z/OS Unicode Conversion Services

You must customize z/OS Unicode support. See Appendix A of *DB2 Installation Guide* and *z/OS Support for Unicode: Using Conversion Services* for more information.

## CCSIDs in DB2

DB2 uses CCSIDs to describe data that is stored in the DB2 subsystem. Beginning in DB2 Version 2 Release 3, you needed to specify the correct CCSID set for the DB2 subsystem to ensure that the appropriate conversion would be carried out for distributed systems that access DB2. These CCSIDs are stored in DSNHDECP.

Beginning in DB2 Version 5, you could store data in ASCII format. You could specify the CCSID ASCII clause in the CREATE statement to define ASCII support for tables, table spaces, and databases. In DB2 Version 7 and later, you can store data in the EBCDIC, ASCII, or Unicode format. In DB2 Version 8, you can compare data in compatible encoding schemes. See Appendix A of *DB2 Installation Guide* for more details about compatible encoding schemes.

You can specify a CCSID as the encoding scheme at any of the following levels:
- System
- Database
- Table space
- Table
- Procedure or function

In addition, you can specify a CCSID as the encoding scheme at the application level or host variable level.

## Using Unicode with DB2 Version 8 utilities

In DB2 Version 8, You can write utility control statements in either EBCIDIC or UNICODE. In DB2 Version 7, you can write utility control statements only in EBCDIC.

The DB2 utilities stored procedures DSNUTILS and DSNUTILU allow you to invoke DB2 utilities from an application program. The stored procedure DSNUTILU accepts utility control statements in Unicode.

The stand-alone utility DSN1PRNT supports the keyword UNICODE. This keyword specifies that the print output is to be displayed in Unicode.

The stand-alone utility DSN1COPY supports the keyword UNICODE. You can use the UNICODE keyword in conjunction with the PRINT option to specify that the output is to be displayed in Unicode.

The LOAD utility supports the keyword UNICODE. This keyword specifies that the input data is in Unicode. If you specify UNICODE with the CONTINUEIF, DEFAULTIF, NULLIF, or WHEN options and the input data is not in the same encoding scheme as the utility control statement, you must write the condition in hexadecimal string form. You cannot use character string form.

The UNLOAD utility supports the keyword UNICODE. This keyword specifies that the output data is in Unicode. If you specify the UNICODE keyword and the data is not in Unicode, the UNLOAD utility converts the data to Unicode.

**Recommendation:** If you use the cross-loader function to transfer delimited files between operating systems or z/OS systems, or z/OS systems that use different EBCDIC or ASCII CCSIDs, use Unicode as the encoding scheme for the delimited file. The use of Unicode helps avoid CCSID conversion problems.

## Specifying locales

Rules for uppercase and lowercase usage vary according to language and country. A *locale* defines the subset of a user's environment that depends on language and cultural conventions. DB2 uses the information that is associated with a locale to execute UPPER, LOWER, and TRANSLATE functions in a culturally correct manner. A locale consists of two components: the first component represents a specific language and country, and the second component is a CCSID.

**Example:** In the locale, Fr_CA.IBM-1047, Fr_CA represents the language and country (French Canadian), and IBM-1047 is the associated CCSID.

The symbol for euro currency is supported through the modifier @EURO.

**Example:** To display results in euro dollars instead of French Francs, specify Fr_FR@EURO.

For a complete list of locales, see *z/OS C/C++ Programming Guide*.

# Chapter 3. How Unicode works with DB2 UDB for z/OS

Even if you do not intend to use Unicode as the encoding scheme for your data, you should be aware of its impact on your DB2 subsystem. In DB2 Version 8, the catalog is converted to Unicode and all parsing is done in Unicode.

## Application encoding

Version 7 introduced support for encoding schemes at the application level. If your application does not specify an encoding scheme, the system default encoding scheme determines the encoding scheme for the application.

At bind time, you can specify the application encoding scheme. The default value is specified on installation panel DSNTIPF. Specifying the encoding scheme at bind time affects static SQL. The use of a special register, which is initialized with the bind option, affects dynamic SQL. You can use DECLARE VARIABLE statements or a CCSID override statement to override the bind option or special register.

**Important:** The bind option is ignored when packages are executed remotely.

**Exception:** The bind option is not ignored for SET statement processing when a package is executed remotely.

**Example:** Assume that the package MY_PACK is bound with the option APPLICATION ENCODING(UNICODE). DB2 assumes that all character input and output host variables for static SQL statements are encoded using CCSID 1208. DB2 assumes that all graphic input and output host variables for static SQL statements are encoded using CCSID 1200. The initial value for the application encoding special register is 1208.

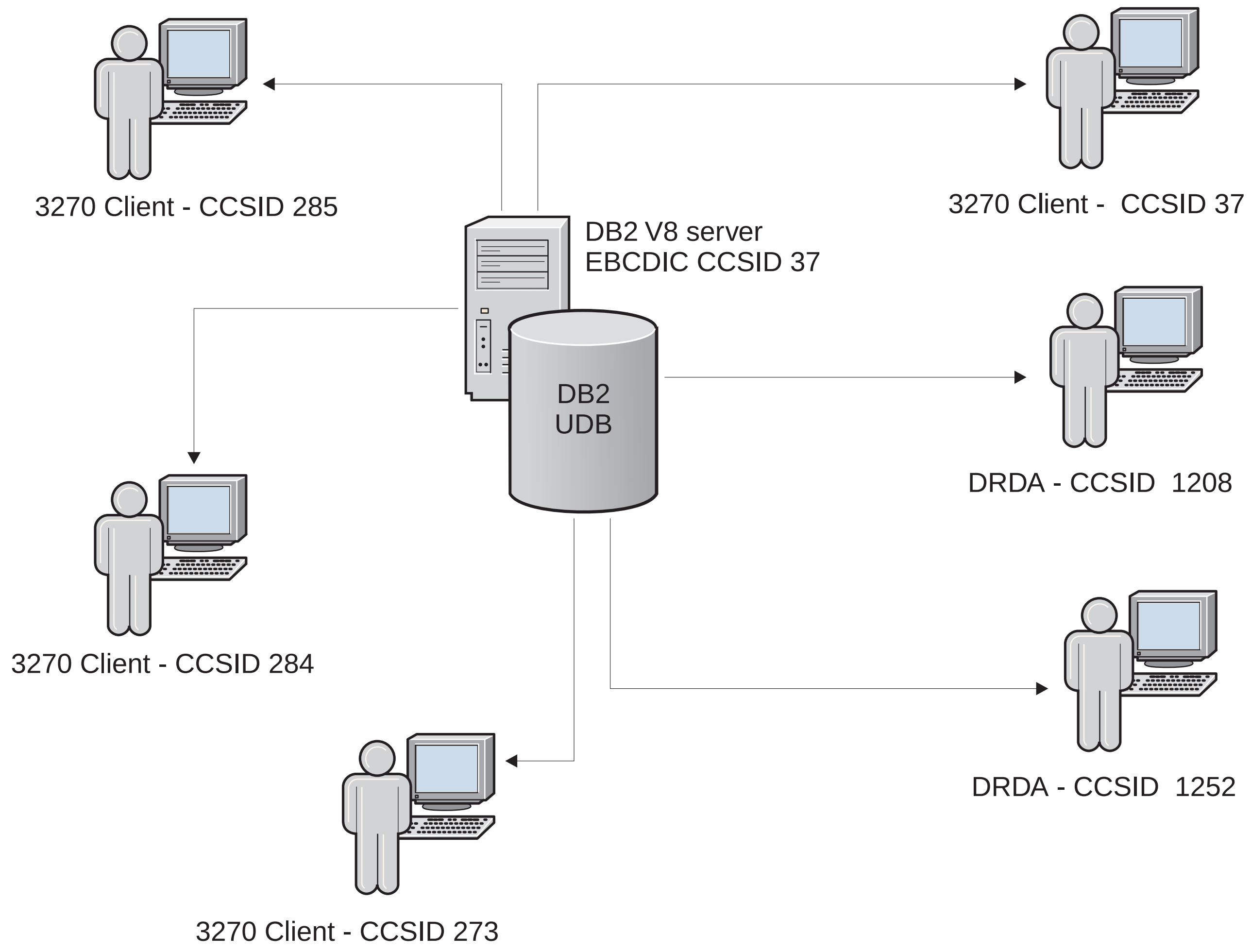

*Figure 1. Example of application encoding*

**Example:** Your DB2 UDB for z/OS Version 8 subsystem is located in the United States. You have users around the world that connect to this subsystem. The users that use DRDA do not need to use the application encoding bind option to handle CCSID conversions because DRDA handles all conversions. The users that use 3270 terminal emulators have set up their emulators ato use a CCSID that corresponds to the country in which they reside. In this example, you could bind an application with ENCODING(37), ENCODING(273), ENCODING(284), and ENCODING(285).

## Version 7-specific Unicode characteristics

DB2 Version 7 introduced Unicode support.

Application encoding information is stored in the following locations:
- SYSIBM.SYSPACKAGE
- SYSIBM.SYSPLAN

Encoding information is also stored in the ENCODING_SCHEME column of the following system tables:
- SYSIBM.SYSDATATYPES
- SYSIBM.SYSDATABASE
- SYSIBM.SYSPARMS
- SYSIBM.SYSTABLESPACE

- SYSIBM.SYSTABLES

Be aware of the following factors that might affect your use of DB2 Version 7.

## Unicode restrictions

The following restrictions exist in Version 7 Unicode support:

- You cannot use incompatible encoding schemes in a single SQL statement. For more information about compatible encoding schemes, see Appendix A of *DB2 Installation Guide*.
- Object names must be representable in your system EBCDIC CCSID.
- Literal values must be representable in your system EBCDIC CCSID.

Some other restrictions might affect Unicode data in Version 7. They include the following restrictions:

- Predicates can compare only 255 bytes.
- The index key size is 255 bytes.
- String literals are 255 bytes.

If you convert your data from EBCDIC or ASCII to Unicode, be aware that Unicode characters can be two to three times the size of EBCDIC or ASCII characters. Thus, if your data is converted to Unicode, your data might be larger than the maximum lengths that are listed above.

## EBCDIC catalog and parsing

In Version 7, the catalog is encoded in the default system EBCDIC CCSID. You must ensure that the names of the following objects can be converted from the CCSID of your application to the default system EBCDIC CCSID:

- Database
- Table space
- Index space
- External object

**Recommendation:** If your DB2 subsystem has users that use different CCSIDs, choose only identifiers such as table names and column names that are represented on all clients that access the DB2 subsystem.

All parsing is done in EBCDIC. DB2 converts data from ASCII or Unicode to the EBCDIC system CCSID, which might result in loss of data. If the CCSID of the statement is ASCII or Unicode, DB2 converts the result to ASCII or Unicode. You must ensure that literal values can be converted from the CCSID that you have chosen to EBCDIC. Use host variables or parameter markers if a literal value cannot be converted to the system EBCDIC CCSID.

## Version 8-specific Unicode characteristics

During DB2 Version 8 enabling-new-function mode processing, DB2 converts the DB2 catalog to Unicode UTF-8. The conversion of the DB2 catalog to Unicode might affect the results of queries against the catalog. String columns in the DB2 catalog are expanded to VARCHAR(128), which allows for easier porting of applications that run on other operating systems. See *DB2 Installation Guide* for more information about the changes to the catalog.

Encoding information is stored in the following locations:

- SYSIBM.SYSCOLUMNS
- SYSIBM.SYSPACKAGE

- SYSIBM.SYSPLAN
- SYSIBM.SYSROUTINES
- SYSIBM.SYSTABLES

Encoding information is also stored in the ENCODING_SCHEME column of the following system tables:

- SYSIBM.SYSDATATYPES
- SYSIBM.SYSDATABASE
- SYSIBM.SYSPARMS
- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSTABLES

The following information explains factors that might affect your migration to DB2 Version 8.

## DB2 internal processing is done in Unicode

The Version 8 precompiler parses the application program in Unicode. If you set the NEWFUN precompiler parameter to YES, the program can use new Version 8 functions. In addition, setting the NEWFUN parameter to YES results in Unicode DBRMs and Unicode statements in the DB2 catalog. If you set the NEWFUN precompiler parameter to NO, the program cannot use new Version 8 functions, and the DBRMs and statements in the catalog are encoded as EBCDIC. If you set the NEWFUN parameter to NO, the precompiler still parses SQL in Unicode.

The following internal processing is done in Unicode:

- The DBRM is parsed in Unicode after you convert your DB2 subsystem to new-function mode.
- SQL statements are parsed in Unicode. Statement text is converted to Unicode if it is not in Unicode.
- Literals in the parse tree are encoded in Unicode.
- Utilities have the option to parse in Unicode.
- Source code is converted to Unicode before being processed by the precompiler, and the source code is parsed in Unicode.
- All catalog access is done in Unicode.
- Within DB2, all authorization is in Unicode. Externally, authorization is in EBCDIC.
- The names of plans and packages are stored in Unicode.
- Special registers are stored in Unicode.
- Output from traces can be stored in Unicode.

## Unicode sorting sequences are different than EBCDIC sorting sequence

The collation sequence for Unicode is different than the sequence that is used for EBCDIC. In Unicode, numeric characters are sorted before alphabetic characters. In EBCDIC, alphabetic characters are sorted before numeric characters. Therefore, your queries against the DB2 catalog might return results in a different order than in DB2 Version 7. GROUP BY, range predicates such as BETWEEN, and functions such as MIN and MAX might also return different answer sets than those that are returned in Version 7.

**Example:** Assume that the NAME column in SYSIBM.SYSTABLES contains the following values: TEST1, TEST2, TEST3, TESTA, TESTB, and TESTC. In Version 7 or Version 8 compatibility mode, you issue the following SQL query:

```
SELECT NAME FROM SYSIBM.SYSTABLES
ORDER BY NAME
```

DB2 returns the following result:

```
TESTA
TESTB
TESTC
TEST1
TEST2
TEST3
```

After you convert your DB2 subsystem to Version 8 new-function mode and issue the same SQL query, DB2 returns the following result:

```
TEST1
TEST2
TEST3
TESTA
TESTB
TESTC
```

**Example:** In Version 7 or Version 8 compatibility mode, you issue the following SQL query:

```
SELECT * FROM SYSIBM.SYSTABLES
WHERE NAME BETWEEN 'TEST2' AND 'TESTB'
ORDER BY NAME
```

DB2 returns the following result:

```
TESTC
TEST1
```

After you convert your DB2 subsystem to Version 8 new-function mode and issue the same SQL query, DB2 returns the following result:

```
TEST3
TESTA
```

In DB2 Version 8 new-function mode, you can simulate the behavior of the ORDER BY clause in DB2 Version 7 and DB2 Version 8 compatibility mode by using the CAST function on your SELECT statement and the ORDER BY clause. Doing so might degrade performance because of the cost of conversion and an index cannot be used to support the ORDER BY clause.

**Example:** In DB2 Version 8 new-function mode, issue the following SQL query:

```
SELECT CAST(NAME AS CCSID EBCDIC) AS E.NAME
FROM SYSIBM.SYSTABLES
ORDER BY NAME
```

DB2 returns the following result:

```
TESTA
TESTB
TESTC
TEST1
TEST2
TEST3
```

## CCSIDs in DSNHDECP must be valid

All CCSIDs in the DSNHDECP module must be valid. During startup processing, if DB2 detects invalid CCSID values, DB2 issues a message and terminates. A list of valid CCSIDs is available in Appendix A of *DB2 Installation Guide*.

## DB2 Version 8 precompiler parses SQL in Unicode

In Version 8, the DB2 precompiler parses application programs in Unicode. In addition, the precompiler has the following new input parameters:

- The NEWFUN parameter specifies whether the program can use new DB2 Version 8 functions. When you specify YES, Unicode DBRMs and Unicode statements are written to the DB2 catalog. If you specify NO, DBRMs and statements are written to the DB2 catalog in EBCDIC.
- The CCSID parameter specifies the CCSID of the source program. This ensures that SQL statements and literal values can be parsed correctly. You should use the CCSID parameter with the application encoding bind option to ensure that the appropriate conversions are performed. The application encoding bind option specifies the CCSID of literal data and host variables in the application.

## Multiple CCSIDs can be referenced in the same SQL statement

After DB2 has entered enabling-new-function mode, multiple CCSIDs can be referenced from the same SQL statement. In previous versions of DB2, an error was returned if you attempted a JOIN or UNION that referenced multiple CCSIDs. This restriction is removed in Version 8 to support table objects (tables, views, temporary tables, query tables, and user-defined functions) with different CCSID sets that are to be referenced in a statement.

**Example:** Assume that EBCDICTABLE is encoded in EBCDIC, and the host variables are encoded in the application encoding scheme. Issue the following SQL statement:

```
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
```

The result of this statement contains multiple CCSIDs.

## ODBC for z/OS has been enhanced for Unicode support

The following DB2 UDB for z/OS ODBC elements support Unicode:

- A new initialization keyword CURRENTAPPENSCH (in the INI file) lets you specify
  - the current encoding scheme (EBCDIC, ASCII, or Unicode) that the ODBC driver is to use for input and output host variable data
  - all character string arguments of the ODBC APIs

  When you set this keyword to UNICODE, generic ODBC APIs support UTF-8 string arguments, and application variables with the symbolic C data type SQL_C_CHAR supports UTF-8 data.
- New APIs with the suffix W, called *wide APIs*, are introduced to support UTF-16 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 Unicode UTF-16 string arguments only. The equivalent wide API for the SQLConnect() function call is SQLConnectW().
- The generic APIs, for example SQLColumnPrivileges, accept UTF-8 string arguments and return all character string data in the result set in UTF-8 encoding scheme.
- The new SQL_C_WCHAR data type supports UTF-16 data.

- The additional SQLGetInfo() attributes, such as SQL_ASCII_SCCSID, let you query the CCSID settings of the DB2 subsystem in each encoding scheme.

# Appendix A. Relationship between a string and its encoding scheme

An encoding scheme and a CCSID are attributes of strings, just as length is an attribute of strings. All values of the same string column have the same encoding scheme and CCSID.

Every string has an encoding scheme and a CCSID that identifies the manner in which the characters in the string are encoded. Strings can be encoded in ASCII, EBCDIC, or Unicode.

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. Table 1 describes the rules for determining the CCSID that is associated with a string value. Use the Type 1 rules when the SQL statement meets these conditions:

- The SQL statement operates with a single set of CCSIDs (SBCS, mixed, and graphic). An SQL statement that does not contain any of the following items operates with a single set of CCSIDs:
  - References to columns from multiple tables or views that are defined with CCSIDs from more than one set of CCSIDs (SBCS, mixed, and graphic)
  - Graphic hexadecimal string constants
  - Unicode hexadecimal string constants
  - References to the XMLCLOB built-in function
  - CAST specifications with a CCSID clause
  - User-defined table functions
- The SQL statement is not one of the following statements:
  - CALL statement
  - SET assignment statement
  - VALUE statement
  - VALUES INTO statement

*Table 1. Rules for determining the CCSID that is associated with string data*

| Source of the string data | Type 1 rules | Type 2 rules |
|---|---|---|
| String constant | If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the string constant.<br><br>Otherwise, the default EBCDIC encoding scheme is used for the string constant.<br><br>The CCSID is the appropriate character string CCSID for the encoding scheme. | The CCSID is the appropriate character string CCSID for the application encoding scheme.[1] |

*Table 1. Rules for determining the CCSID that is associated with string data (continued)*

| Source of the string data | Type 1 rules | Type 2 rules |
|---|---|---|
| Hexadecimal string constant (X'...') | If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the string constant.<br><br>Otherwise, the default EBCDIC encoding scheme is used for the string constant.<br><br>The CCSID is the appropriate graphic string CCSID for the encoding scheme. | The CCSID is the appropriate character string CCSID for the application encoding scheme.[1] |
| Graphic string constant (G'...') | If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the graphic string constant.<br><br>Otherwise, the default EBCDIC encoding scheme is used for the graphic string constant.<br><br>The CCSID is the graphic string CCSID for the encoding scheme. | The CCSID is the graphic string CCSID for the application encoding scheme.[1] |
| Graphic hexadecimal constant (GX'...') | Not applicable. | The CCSID is the graphic string CCSID for the application encoding scheme, which must be ASCII or EBCDIC. |
| Hexadecimal Unicode string constant (UX'....') | Not applicable. | The CCSID is 1200 (UTF-16). |
| Special register | If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the special register.<br><br>Otherwise, the default EBCDIC encoding scheme is used for the special register.<br><br>The CCSID is the appropriate character string CCSID for the encoding scheme. | The CCSID is the appropriate CCSID for the application encoding scheme.[1] |
| Column of a table | The CCSID is the CCSID that is associated with the column of the table. | The CCSID is the CCSID that is associated with the column of the table. |
| Column of a view | The CCSID is the CCSID of the column of the result table of the fullselect of the view definition. | The CCSID is the CCSID of the column of the result table of the fullselect of the view definition. |
| Expression | The CCSID is the CCSID of the result of the expression. | The CCSID is the CCSID of the result of the expression. |

*Table 1. Rules for determining the CCSID that is associated with string data (continued)*

| **Source of the string data** | **Type 1 rules** | **Type 2 rules** |
|---|---|---|
| Result of a built-in function | If the description of the function in *DB2 SQL Reference* indicates what the CCSID of the result is, the CCSID is that CCSID.<br><br>Otherwise, if the description of the function refers to this table for the CCSID, the CCSID is the appropriate CCSID of the CCSID set that is used by the statement for the data type of the result. | If the description of the function in *DB2 SQL Reference* indicates what the CCSID of the result is, the CCSID is that CCSID.<br><br>Otherwise, if the description of the function refers to this table for the CCSID, the CCSID is the appropriate CCSID of the CCSID set that is used by the statement for the data type of the result.[1] |
| Parameter of a user-defined routine | The CCSID is the CCSID that was determined when the function or procedure was created. | The CCSID is the CCSID that was determined when the function or procedure was created. |
| The expression in the RETURN statement of a CREATE statement for a user-defined SQL scalar function | If the expression in the RETURN statement is string data, the encoding scheme is the same as for the parameters of the function. The CCSID is determined from the encoding scheme and the attributes of the data. | The CCSID is determined from the CCSID of the result of the expression that is specified in the RETURN statement. |
| String host variable | If the statement references a table or view, the encoding scheme of that table or view determines the encoding scheme for the host variable.<br><br>Otherwise, the default EBCDIC encoding scheme is used for the host variable.<br><br>The CCSID is the appropriate CCSID for the data type of the host variable. | At bind time, the CCSID is the appropriate CCSID for the data type of the host variable for the application encoding scheme.<br><br>At run time, the CCSID is specified in the DECLARE VARIABLE statement, or as an override in the SQLDA. Otherwise, the CCSID is the appropriate CCSID for the data type of the host variable for the application encoding scheme. |

**Note:** [1]If the context is within a check constraint or trigger package, the CCSID is the appropriate CCSID for Unicode instead of for the application encoding scheme.

# Appendix B. Common conversions for DB2 UDB for z/OS

DB2 Version 8 requires you to provide a DSNHDECP module that specifies valid, non-zero CCSIDs for single-byte character sets (SBCS) for both EBCDIC and ASCII. For Chinese, Japanese, and Korean, you must also specify valid, non-zero CCSIDs for mixed-byte character sets (MBCS) and double-byte character sets (DBCS).

DB2 uses the same CCSIDs for Unicode, regardless of language. The Unicode CCSIDs that DB2 uses are:

| | |
|---|---|
| **367** | SBCS data |
| **1208** | MBCS data, Unicode UTF-8 |
| **1200** | DBCS data, Unicode UTF-16 |

CCSID 1200 supports the same group of characters as CCSID 1208, so they are compatible and can convert from any EBCDIC or ASCII CCSIDs.

All DB2 Version 8 subsystems require the use of z/OS Unicode Services and appropriate conversion definitions to perform most Unicode conversions. See *z/OS Support for Unicode: Using Conversion Services* for more information.

## Basic conversions

When you plan your z/OS Unicode Services setup, begin with a set of basic conversions between the CCSIDs for SBCS, DBCS, and MBCS. Use the following conversions:

```
CONVERSION 367, 1200, ER;
CONVERSION 367, 1208, ER;
CONVERSION 1200, 367, ER;
CONVERSION 1200, 1208, ER;
CONVERSION 1208, 367, ER;
CONVERSION 1208, 1200, ER;
```

If you use the samples that are provided with DB2, define the following conversions also:

```
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;
```

**Recommendation:** 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, you must specify the following conversions:

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

In these CONVERSION statements, *sccsid* is the EBCDIC SBCS CCSID that is specified in your DSNHDECP module.

**Recommendation:** For completeness, define the following conversions between the EBCDIC SBCS CCSID that is defined in your DSNHDECP module and CCSID 37, and between the EBCDIC SBCS CCSID that is defined in your DSNHDECP module and CCSID 1047.

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

## Additional conversions for Chinese, Japanese, and Korean

For Chinese, Japanese, and Korean character sets, you need to specify several conversions in addition to the basic conversions.

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

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

In these conversion statements, *mccsid* is your EBCDIC MBCS CCSID.

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

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

In these conversion statements, *gccsid* is your EBCDIC DBCS CCSID.

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

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

In these conversion statements, *asccsid* is your ASCII SBCS CCSID.

**Recommendation:** For completeness, specify conversions between your ASCII SBCS CCSID and CCSID 37, and between your ASCII SBCS CCSID and CCSID 1047. Use the following CONVERSION statements:

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

Specify conversions between your ASCII MBCS CCSID and each Unicode CCSID:

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

In these conversion statements, *amccsid* is your ASCII MBCS CCSID.

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

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

In these conversion statements, *agccsid* is your ASCII DBCS CCSID.

If your DSNHDECP module specifies an EBCDIC SBCS CCSID other than CCSID 37 or CCSID 1047, you need additional conversions:

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

**Recommendation:** Also add a conversion between your MCCSID and your AMCCSID and a conversion between your GCCSID and your AGCCSID. Use the following conversion statements:

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

# Appendix C. DD names passed to CUNJIUTL

The DD names that are passed to CUNJIUTL are described in this appendix.

## SYSPRINT

SYSPRINT is a listing that shows the processed setups and error messages, if applicable.

## TABIN

TABIN provides conversion tables for character conversion and case conversion. They are supplied by IBM. The conversion image transforms the conversion tables into an internal format and stores them in the conversion image.

## SYSIMG

SYSIMG supplies output that is a single image of the entire conversion environment. The conversion image is built according to the specification in the SYSIN DD name. The conversion image resides in either a sequential data set or a member of a partitioned data set with a fixed-block 80-byte format.

## SYSIN

Two types of statements are recognized in this DD statement: case conversion, which is identified by the CASE control statement, and character conversion, which is identified by the CONVERSION control statement.

### CASE control statement

Case conversion is defined as converting Unicode characters (for example, UTF-8) to their uppercase equivalent or their lowercase equivalent. When CASE NORMAL is specified, basic case conversion is provided. This basic case conversion is based on the UnicodeData.txt file that is provided by the Unicode consortium. It does not include special casing as described in the SpecialCasing.txt file that is provided by the Unicode consortium. Special casing typically includes characters that have significant differences in the case-based appearance. For example, the German “hard S”, which appears as a flat B, appears as “SS” in uppercase German text. Because DB2 does not use the case conversion service, you do not need to specify a conversion.

If you want to use the UPPER or LOWER built-in functions to process Unicode data, and if you want to perform case conversions on characters other than A-Z and a-z, you must provide additional control statements to the z/OS Unicode Support Services. For more information, see *DB2 SQL Reference* and *z/OS Support for Unicode: Using Conversion Services.*

**Example:** Assume that your DB2 UDB for z/OS subsystem has defined CCSID 37 for the system EBCDIC CCSID and CCSID 819 for the system ASCII CCSID. Assume that your DB2 UDB for z/OS subsystem has the following conversions defined:

```
CONVERSION 37,367,ER;
CONVERSION 37,1208,ER;
CONVERSION 37,1200,ER;
CONVERSION 367,37,ER;
CONVERSION 1208,37,ER;
CONVERSION 1200,37,ER;
CONVERSION 819,367,ER;
CONVERSION 819,1208,ER;
CONVERSION 819,1200,ER;
CONVERSION 367,819,ER;
CONVERSION 1208,819,ER;
CONVERSION 1200,819,ER;
```

For DB2 to correctly perform additional case conversions, modify the conversion image to the following statements:

```
CASE NORMAL; /* Normal Casing */
CASE SPECIAL; /* Additional Locale Independent casing */
CASE LOCALE; /* Locale dependent casing */
CONVERSION 37,367,ER;
CONVERSION 37,1208,ER;
CONVERSION 37,1200,ER;
CONVERSION 367,37,ER;
CONVERSION 1208,37,ER;
CONVERSION 1200,37,ER;
CONVERSION 819,367,ER;
CONVERSION 819,1208,ER;
CONVERSION 819,1200,ER;
CONVERSION 367,819,ER;
CONVERSION 1208,819,ER;
CONVERSION 1200,819,ER;
```

These additional statements provide the necessary infrastructure for the UPPER and LOWER functions to process Unicode data according to the Unicode standard.

## CONVERSION control statement

Character conversion is also referred to as conversion between specified CCSIDs. An application such as DB2 invokes the CUNLCNV function to convert characters between the specified code pages. You must identify the conversions that are possible on the CONVERSION control statement.

Specify CONVERSION statements for DB2 as follows:

```
CONVERSION xxx,yyy,ER;
CONVERSION yyy,xxx,ER;
```

Many code page conversions are possible. They are documented in *z/OS Support for Unicode: Using Conversion Services*. However, when identifying the conversion that DB2 is to use, you need to be concerned only with conversion for the national languages that you use and with conversion from these code pages to and from all Unicode CCSIDs.

**Example:** If you use an EBCDIC CCSID of 37 and an ASCII CCSID of 819, use the following conversions:

```
CONVERSION 37,367,ER;
CONVERSION 37,1208,ER;
CONVERSION 37,1200,ER;
CONVERSION 367,37,ER;
CONVERSION 1208,37,ER;
CONVERSION 1200,37,ER;

CONVERSION 819,367,ER;
CONVERSION 819,1208,ER;
CONVERSION 819,1200,ER;
CONVERSION 367,819,ER;
CONVERSION 1208,819,ER;
CONVERSION 1200,819,ER;
```

Multiple conversion tables might be available for converting one CCSID to another. You can use a technique search order (not shown in the preceding example) to specify which table should be used. The technique search order consists of up to eight technique characters. If you specify more than one technique character, the image generator tries to find a matching table for the leftmost technique character in the sequence of the technique search order. If one is not found, the search continues with the second one, and so on. Especially for mixed conversion, use more than one technique character because one of the subconversions might exist only in round-trip mode, and one might exist only in an enforced subset. In this case, a technique search order of ’RE’ or ’ER’ would be required. Technique search order is optional. If you do not specify a technique search order, RECLM is used.

In the example above, technique search order is not specified on any of the CONVERSION statements. Therefore, they are logically equivalent to the following specifications:

```
CONVERSION 1047,850,RECLM; /* EBCDIC -> ASCII */
CONVERSION 850,1047,RECLM; /* ASCII -> EBCDIC */
```

The important technique characters for DB2 are E (enforced subset) and R (round-trip). Enforced subset conversions map only those characters from one CCSID to another that have a corresponding character in the second CCSID. All other characters are replaced by a substitution character. Round-trip conversions between two CCSIDs assure that all characters that make the “round trip” arrive as they were originally, even if the receiving CCSID does not support a given character. Round-trip conversions ensure that code points that are converted from CCSID A to CCSID B, and back to CCSID A are preserved, even if CCSID B is not capable of representing these code points.

# Appendix D. SYSIBM.SYSSTRINGS

This appendix contains a description of the columns in the SYSIBM.SYSSTRINGS catalog table.

## Columns in SYSIBM.SYSSTRINGS

The catalog table SYSIBM.SYSSTRINGS contains the following columns:

- INCCSID
- OUTCCSID
- TRANSTYPE
- ERRORBYTE
- SUBBYTE
- TRANSPROC
- IBMREQD

### INCCSID

The entry in column INCCSID defines the source CCSID of a character conversion. This value must be in the range of 1 to 65533.

**Restriction:** For any given row, the INCCSID and OUTCCSID columns cannot contain the same value.

### OUTCCSID

The entry in column OUTCCSID defines the target CCSID of a character conversion. This value must be in the range of 1 to 65533.

**Restriction:** For any given row, the INCCSID and OUTCCSID columns cannot contain the same value.

### TRANSTYPE

TRANSTYPE determines the type of conversion. The entry in this column must be one of the following two-character strings:

| | |
|---|---|
| **SS** | SBCS data to SBCS data |
| **SM** | SBCS data to EBCDIC MIXED data |
| **MS** | EBCDIC MIXED data to SBCS (EBCDIC and ASCII) data |
| **PS** | ASCII MIXED data to SBCS (EBCDIC and ASCII) data |
| **GG** | GRAPHIC data to GRAPHIC data |
| **PM** | ASCII MIXED data to EBCDIC MIXED data |
| **MM** | EBCDIC MIXED data to EBCDIC MIXED data |
| **MP** | EBCDIC MIXED to ASCII MIXED data |
| **PP** | ASCII MIXED to ASCII MIXED data |
| **SP** | SBCS (ASCII and EBCDIC) to ASCII MIXED data |

### ERRORBYTE

The entry in the ERRORBYTE column specifies the byte that is used in the conversion table (specified in the TRANSTAB column) as an error indicator. If you specify a null value in this column, no error indicator is established.

If you enter a non-null value in this column, you cannot use the same value in the SUBBYTE column.

**Example:** If ERRORBYTE is X'3E', that byte is used in the conversion table as a substitute for code points that map to X'3E'.

### SUBBYTE

The value in the SUBBYTE column specifies the byte that is used in the conversion table (TRANSTAB) as a substitution character. If you specify a null value in this column, no substitution character is established.

If you enter a non-null value in this column, you cannot use the same value in the ERRORBYTE column.

**Example:** If SUBBYTE is X'3F', that byte is used in the conversion table as a substitute for code points that map to X'3F'. DB2 issues a warning when a code point maps to the value of SUBBYTE.

### TRANSPROC

The entry in the TRANSPROC column specifies the name of a module or a blank string. The value in this column must either be blank or contain a string that conforms to the rules for z/OS program names.

If the value that is specified in the IBMREQD column is N, a non-blank value of TRANSPROC is the name of a user-provided conversion procedure. If the value specified in the IBMREQD column is Y, a non-blank value of TRANSPROC is the name of a DB2 module that contains DBCS conversion tables.

### IBMREQD

A value of Y indicates that the row is provided by IBM. Rows that are provided by IBM cannot be updated or deleted.

A value of N indicates that the row is inserted by the user. Rows with IBMREQD=N can be inserted, updated, and deleted.

### TRANSTAB

TRANSTAB contains 256-byte conversion table or an empty string. The length that is specified in the TRANSTAB column must be either 0 or 256.

## Description of entries in SYSIBM.SYSSTRINGS

Each row of SYSSTRINGS contains information about the conversion of character strings from the coded character set that is identified by INCCSID to the coded character set that is identified by OUTCCSID. The conversion function is automatically invoked when a conversion is required from the coded character set that is identified by the INCCSID column to the coded character set that is identified by the OUTCCSID column.

**Example:** The row of SYSSTRINGS in which the value of INCCSID is 500 and the value of OUTCCSID is 37 describes the conversion from CCSID 500 to CCSID 37. The row in which the value of INCCSID is 37 and the value of OUTCCSID is 500 describes the conversion from CCSID 37 to CCSID 500.

The same pair of CCSIDs can be in two rows, if one is in an IBM-supplied row and the other is in a user-provided row. In this case, the user-provided row is used for the character conversion.

# Notices

This information was developed for products and services offered in the U.S.A. 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
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** 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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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

## Trademarks

| | |
|---|---|
| CICS | IBM |
| DB2 | IMS |
| DB2 Universal Database | Language Environment |
| Distributed Relational Database Architecture | QMF |
| DRDA | z/OS |

# Bibliography

***DB2 UDB for z/OS***

- *DB2 Installation Guide*
- *DB2 SQL Reference*

**Character Data Representation Architecture**

- *Character Data Representation Architecture Overview*, GC09-2207
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

**Unicode standard**

Information about Unicode, the Unicode consortium, the Unicode standard, and standards conformance requirements is available at www.unicode.org

**z/OS**

- *z/OS C/C++ Programming Guide*, SC09-4765
- *z/OS Support for Unicode: Using Conversion Services* , SA22-7649

Printed in USA