All for One and One for All - Introducing Unicode (Part 1)

Wherever you are in the world and whatever your own code page requirements, DB2 for z/OS Version 8 will bring Unicode into your world! There are fundamental changes on the horizon which build on Version 7's ability to create Unicode tables, lifting the current restrictions and significantly enhancing DB2's Unicode solution. These changes include the conversion of the DB2 catalog from EBCDIC to Unicode and the introduction of Unicode parsing, impacting everyone to some degree. In addition, V8 provides significant new functionality in the code page area, including the ability for a single SQL statement to access a combination of Unicode, EBCDIC and ASCII tables.

The objective of this two-part article is to explain the changes in V8 and help you understand the impact they will have in order to assist with preparation and planning. It will include explanations of the features that can be used to minimize any disruption.

In this issue, we will cover the following topics to give an understanding of Unicode and DB2's code page support so far:

- Back to basics - some code page terminology explained
- So what is Unicode?
- The evolving story of DB2's code page support
- What's new in Version 8.

The article will continue in an upcoming issue, describing the impact of the V8 changes in more depth as follows:

- How the DB2 catalog changes in V8 will affect you
- Program preparation in V8
- Storing application data and writing applications in Unicode.

Let's destroy some myths!

Myth 1: In V8, you will have to convert your application's stored data to Unicode. V8 enhances the infrastructure to support storing and retrieving data in Unicode format and this includes conversion of the DB2 catalog to Unicode. However, the intention is not to force user data to be stored as Unicode, but to provide a choice. IBM expects most customers to keep the majority of their data in its current EBCDIC or ASCII format for some time. As more customers develop the need to store international data, more will convert to using Unicode.

Myth 2: Converting to Unicode will always double your storage requirement.

There is a common misunderstanding that Unicode doubles the size of the storage needed for your data. This misconception is derived from the original Unicode transformation format of UCS-2 which stores every character as two bytes. However, additional Unicode transformation formats have been devised, and these include UTF-8 and UTF-16, which the DB2 Family uses to store character and graphic data respectively.
In UTF-8, the first 127 code points are the same as ASCII, with one byte being used for characters such as A-Z, a-z and 0-9. (Code points are defined below in the Terminology section.) Other characters are stored as one to four bytes, with accented characters often taking two bytes and Far Eastern characters taking three to four bytes. Therefore, the actual storage requirement for your system will depend on the nature of your data.

Myth 3: It doesn't affect me!
Everyone WILL be impacted by the changes in V8 regardless of whether user data is stored as Unicode. The rest of the paper will describe this in detail, but a summary of the key areas follows:

1) In V8, SQL will always be parsed in Unicode, whether running in Compatibility Mode or New Function Mode. In addition, when specifying the new precompiler parameter NEWFUN=YES, the DBRMs and bound statements in the DB2 catalog will be stored as Unicode and may not be readable in the normal way (e.g. via ISPF browse).

2) The conversion of the DB2 catalog to Unicode may impact the result of queries against it. Unicode has a different collating sequence from EBCDIC as numbers come before letters, and upper case comes before lower case. This has the potential to impact ORDER BY and range predicates although ranges or sorts within a case or within the numbers may not be affected. In addition to the conversion to Unicode, V8 expands the string columns in the DB2 catalog to Varchar(128) to allow for easier porting of applications which run on other platforms and database management systems.

3) It is important for application developers and DBAs to understand whether applications that access the DB2 catalog are affected and how the precompiler and bind options can be used to ensure transparency from the database CCSID (Coded Character Set Identifier).

Let's go back to basics…

Some code page terminology explained

Code pages have not been an issue or concern for many people up until now, so before we start discussing Unicode and V8, let's step back and recap some terminology.

What are code points?

If we step right back to computing fundamentals, all data is stored as bytes. For example, in our DB2 for z/OS (EBCDIC) systems, we are familiar with the character 'a' being stored as X'81', the character 'A' as X'C1' and the character representation of number '1' as X'F1'.

These byte representations for characters are called code points.

So what are code pages and CCSIDs?

A code page is a set of code points for a particular character set. For example, the code points above are two examples within the EBCDIC code pages.

A CCSID is simply a number to identify a particular code page. For example, North Americans use the US-English code page denoted by a CCSID of 037. Germans use the CCSID 273 that includes code points for specific characters in their language such as letters with umlauts. Other examples include 1252 which is an ASCII CCSID used on the Windows platform and 1208 which represents the Unicode transformation format UTF-8.

Then what is an encoding scheme?

An encoding scheme is a collection of code pages (CCSIDs) for various languages used on a particular computing platform (or group of platforms). For example, the EBCDIC encoding scheme is used on z/OS and iSeries (AS/400) systems. The ASCII encoding scheme is used on Intel-based (Windows) systems and UNIX-based systems.

ASCII stores the character 'A' as X'41', the character 'a' as X'61' and the number '1' as X'31'. This results in a different collating sequence to EBCDIC where lower case is followed by upper case and then numerals.

And a CCSID set?

A CCSID set is comprised of a single byte, a mixed byte and a double byte CCSID, and is specified when installing a DB2 system.

Most languages only use single byte CCSIDs. Therefore the mixed and double byte CCSIDs in the CCSID set don't apply and default to 65534 which is a reserved or dummy CCSID.

The languages which use more than a single byte CCSID are Chinese, Japanese and Korean. These use double byte and mixed character sets due to the range and

complexity of their symbols. (Mixed data is the ability to mix single and double byte characters in CHAR/VARCHAR columns. In EBCDIC, double byte characters are preceded by a "shift in" character and followed by a "shift out" character.)

OK - So what is Unicode?

In this era of globalization, the ability for systems to be able to handle data from around the world is becoming paramount. However, workstations and servers can use different code pages, depending on the native language of the workstation user. In effect the workstation and servers are speaking "different languages" and this makes communication difficult.

For example, if a workstation inserts some data into a DB2 for z/OS system, the data is converted from ASCII to EBCDIC using a conversion table, which maps the code points from the source (ASCII) CCSID to the target (EBCDIC) CCSID.

In addition to a conversion cost, a more serious issue is the potential loss of characters. For example, if a Japanese workstation were inserting data into a European DB2 system, many characters would not have a code point in the CCSID used by DB2. Either the characters must be lost (enforced subset conversions) or DB2 must map them to code points that are not already used (a round trip conversion). The problem with the second option is that another system reading the data will not know about this mapping and may not read the data correctly, perhaps mapping the characters to some of its own characters.

The design objective of Unicode is to avoid these issues by having a single code page that has a code point mapping for every character in the world. The Unicode Consortium has devised a number of Universal Transformation Formats (UTFs) which include unique code points for most current and historical languages, mathematical and scientific symbols, and can be extended as new characters emerge. These UTFs have become widely accepted, being used by technologies such as Java, XML and LDAP.

Many consider Unicode as the foundation for globalization of data and it is becoming a strategic direction for many companies. For example, Microsoft has adopted Unicode with products such as Word by storing data in Unicode and by providing Unicode APIs for ODBC.

A summary of the various Unicode Transformation Formats

There are a number of Unicode Transformation Formats, each of which is designed to handle characters from around the world. The difference between the UTFs is that they are geared towards storing different types of data, having different code points for the characters.

For example, UTF-8 was designed as an Internet encoding transformation format and is used by DB2 for storing character data. Each character is stored as one to four bytes with the first 127 code points the same as ASCII, e.g. A = X'41' and 1 = X'31'

UTF-16 is geared towards storing double byte characters with many of the Chinese, Korean and Japanese characters being stored as two bytes. These same characters can be stored in UTF-8 but use three to four bytes and therefore require extra storage.

These transformation formats are the ones used by DB2 to store character and graphical data respectively. Other transformation formats include UTF-32 which stores each character as four bytes and UCS-2 which has been deprecated in favor of UTF-16.

(Note that Unicode only affects character data or numeric data stored as characters ie. CHAR, VARCHAR, GRAPHIC, VARGRAPHIC, CLOB, DBCLOB. Numeric data stored as as binary, packed or floating point are not affected.)

The History of DB2's code page support

Since DB2 Version 2 Release 3, the systems programmer has needed to specify the correct CCSID set for the system to ensure that the appropriate conversion would be carried out for distributed systems accessing DB2. The CCSIDs are stored in DSNHDECP and, provided that SYSIBM.SYSSTRINGS contained a conversion table for all required code page conversions, there were no problems.

That was the extent of DB2's code page support until V5 introduced the ability to store data in ASCII format to assist SAP R/3 and distributed applications. This could be defined at the table, table space or database level using the CCSID ASCII clause in the CREATE statement.

Recently, Unicode has been evolving. It has become more important to customers that their DB2 systems support more than a single character set. Technologies such as Java and XML, which are founded on Unicode, have become more common, also increasing demand for DB2's Unicode support. Version 7 introduced the ability to store and retrieve Unicode data in a similar way to the ASCII support with a CCSID UNICODE clause on the object definitions. UTF-8 was chosen as the default format for character data columns, with UTF-16 for graphic data columns.

V7 also introduced a new parameter APPLICATION ENCODING SCHEME used to determine the CCSID for the host variable data in static SQL. This could be specified as a bind option for static SQL, defaulting to the subsystem value specified during installation. The value can also be overridden for dynamic SQL by a new APPLICATION ENCODING special register. (Further information on storing and retrieving Unicode data can be found later in part 2 of this article.)

Although a large step forward, there are limitations with this Unicode solution as the DB2 catalog and parser in V7 still use EBCDIC. This prevents Unicode literals and Unicode object names from being specified in SQL. In addition, it is not possible for a single SQL statement to access tables with different CCSIDs, e.g. joins. V8 lifts these restrictions significantly enhancing the Unicode support provided by DB2.

What's new in Version 8?

In order to remove these restrictions, there are some fundamental changes in Version 8, which are summarized as follows:

- Conversion of the data in the DB2 catalog to Unicode UTF-8.
- Two new precompiler options: the CCSID option specifies the CCSID in which the program source code is written. The NEWFUN option denotes whether new functionality can be used in the program and also whether the DBRM and statements in the DB2 catalog are to be stored as Unicode. (Note that the SQL is parsed in Unicode whichever option is used.)

In an upcoming issue we will describe these changes in more detail, explaining the impact they will have and providing assistance in planning for Version 8. In the meantime, please refer to the Solutions Journal Volume 10, Number 1 (2003) in which DB2 for z/OS Version 8 features were described at length.

Summary

I hope that this article has begun to give you some understanding of Unicode and the impact of the enhancements in DB2 for z/OS Version 8. Part 2 will describe the impact in more detail, explaining how the conversion of the DB2 catalog to Unicode will impact you, along with an explanation of the program preparation enhancements.

.

Sarah Ellis
8th September, 2003

Further Information

Unicode Consortium : www.unicode.org
DB2 for z/OS V8 Installation Guide and Application Programming Guide
Section 6.2 of Redbook SG24-6871