Business Intelligence Competency Center (BICC) Core System Documentation:

Encoding & Unicode Considerations

This document describes in detail what you need to know when dealing with Encoding and Unicode in applications built on top of Microsoft solutions. To ensure the highest compatibility, use Unicode in international contexts and apply the best practices discussed in this document.

Main take-outs and Best Practices

The following statements provide guidance regarding Encodings and Unicode in your projects.

The best choice for a code page-specific server is to communicate only with the clients using the same code page. If you must communicate with clients using different code pages, the supported solution is to store your data in Unicode columns. If any one of these options is not feasible, the other alternative is to store the data in binary columns using the binary, varbinary, or varbinary (max) data types.

In international databases, character data must always use Unicode nchar and nvarchar data types in place of their non-Unicode equivalents (char and varchar). Nchar/nvarchar type is the Unicode data type used by Microsoft® SQL Server™, which can store any characters defined by Unicode.org. Char/varchar/text type is always associated with a code page, so the number of supported characters is limited. For example:

- Starting from Windows® 2000, Windows has full Unicode support implemented by using UTF-16 Encoding. Windows APIs take WCHAR* as input which represent a Unicode String.
- Virtual C++ has WCHAR, which is Unicode char type.
- .Net String is Unicode String only encoded in UTF-16.
- Java String is Unicode String only encoded in UTF-16.
- Char type might have a data corruption issue if your client locale is different from the server locale.

The only benefit of using char type is the space saving for single byte code page. You can use the Data Compression feature in SQL Server 2008 if you care about disk space, which can save more space than using char type.

Put N' for your string literal in T-SQL. String literal N'abc' is a Unicode nvarchar string literal. Literal 'ab©' is a varchar literal, it always associates with a code page (string literal always use current database's collation). Suppose char © is not in the code page, you will get a question mark (?) when inserting the literal into a table, even If the column is nvarchar type.

Use nvarchar instead of nchar. The difference between nvarchar and nchar is the storage. A column with nchar(30) type always takes 60 bytes on the disk, even if the value is a single character. The data size for an nvarchar(30) type column is not fixed; it varies row by row or value by value. A single character value takes 2 bytes, and a value with 30 characters long takes 60 bytes to be stored. Another difference between nvarchar and nchar types is performance. Nchar type is always stored in fixed location in every row, which can be retrieved faster than nvarchar type which is stored in different locations for different rows. However, the benefit of less stored space for nvarchar types usually overcomes the cost of locating the value in a row.

If possible, avoid using a column collation which is different from the database's collation. You will have less collation conflicts. If you want a query to use a special collation's sorting rule, use explicitly the collate clause in that query.

Use Windows collations instead of SQL Server collations. The only exception here is the default collation for en_US locale which is sql_latin1_general_cp1_ci_as.

Never store UTF-8 string in varchar types, you will get data corruption.

String comparison always ignores trailing spaces (Unicode U+0020), no matter what collation you are using. LEN function always returns the number of characters excluding trailing spaces. DataLength function returns the storage size in terms of bytes.

Use _BIN2 collation, instead of _BIN collation if you want binary, code page based string comparison behavior.

Globalization standards

The code page issue

In a computer, characters are represented by different patterns of bits being either ON or OFF. There are 8 bits in a byte, and the 8 bits can be turned ON and OFF in 256 different patterns. A program that uses 1 byte to store each character can therefore represent up to 256 different characters by assigning a character to each of the bit patterns.

There are 16 bits in 2 bytes, and 16 bits can be turned ON and OFF in 65,536 unique patterns. A program that uses 2 bytes to represent each character can represent up to 65,536 characters.

Single-byte code pages are definitions of the characters mapped to each of the 256 bit patterns possible in a byte.

Code pages define bit patterns for uppercase and lowercase characters, digits, symbols, and special characters such as !, @, #, or %.

Each European language, such as German or Spanish, has its own single-byte code page. Although the bit patterns used to represent the Latin alphabet characters A through Z are the same for all the code pages, the bit patterns used to represent accented characters such as 'é' and 'á' vary from one code page to the next.

If data is exchanged between computers running different code pages, all character data must be converted from the code page of the sending computer to the code page of the receiving computer. If the source data has extended characters that are not defined in the code page of the receiving computer, data is lost.

When a database serves clients from many different countries, it is difficult to pick a code page for the database that contains all the extended characters required by all the client computers. Also, there is a lot of processing time spent doing the constant conversions from one code page to another.

Single-byte character sets are also inadequate to store all the characters used by many languages. For example, some Asian languages have thousands of characters, so must use two bytes per character.

Double-byte character sets have been defined for these languages. Still, each of these languages have their own code page, and there are difficulties in transferring data from a computer running one double-byte code page to a computer running another.

Standard Organizations

ANSI

The **American National Standards Institute** (www.ansi.org) is an organization that standardizes various areas, both public and private. It is an acronym for American National Standards Institute. ANSI has standardized many areas of computing.

One of the standards in computers was the character set (letters, numbers, and symbols) that a computer can use. This is a standardized encoding table (a code page) covering all uppercase and lowercase English letters, digits, punctuation characters, as well as some special and control characters.

At the very beginning computers could work with 128 different combinations (one bit was reserved for other purposes).

The ASCII code page covered 128 characters, but newer computer systems were able to work with 256 codes and engineers soon noticed that 128 codes are not sufficient for all characters. Even the diacritical marks of the Western European languages could not be covered.

So the standard committees (ANSI, ISO) and computer companies (IBM, Apple, Microsoft) started extending the ASCII code page with various character sets.

The complementary 128 codes had been filled with graphical symbols, mathematical signs, Western European diacritical marks etc. Each organization elaborated its own "standards".

ANSI and Microsoft invented the code page 1252 (ANSI Latin-1), the International Standards Organization (ISO) established the ISO 8859-1 (ISO Latin-1), IBM developed the code page 850 (IBM Latin-1), Apple created the Macintosh Roman character set, etc.

Each Windows system has a default ANSI code page according to system regional settings (932 commonly known as Shift JIS for Japan computers, 1252 commonly known as ANSI Latin-1 for Western European computer.

Unicode Consortium

The Unicode Consortium is a non-profit organization founded to develop, extend and promote use of the Unicode Standard, which specifies the representation of text in modern software products and standards.

The membership of the consortium represents a broad spectrum of corporations and organizations in the computer and information processing industry.

The consortium is supported financially solely through membership dues. Membership in the Unicode Consortium is open to organizations and individuals anywhere in the world who support the Unicode Standard and wish to assist in its extension and implementation.

The Unicode standard is a character coding system designed to support written texts of diverse modern, classical and historical languages. It's based on double-byte character encoding, so it can enumerate 65,536 characters. It's hopefully the "one and only" future standard, and it may solve the "code page soup" problem.

Unicode is compatible with the ISO 10646 standard. The current version 5.2 includes 107,361 coded characters used in written languages of the Americas, Europe, the Middle East, Africa, India, Asia, and Pacifica.

The industry is converging on Unicode for all internationalization. For example: Microsoft Windows Operating Systems have been built on a base of Unicode since Windows NT; AIX, Sun and HP/UX offer Unicode support.

All web standards: HTML, XML, WML etc. are supporting or requiring Unicode. Most versions of Web Browser support Unicode. Sybase, Oracle, SQL, DB2 all offer or are developing Unicode support.

Unicode is the only way of globalization.

UTF-7 encoding: (UTF-7 stands for UCS Transformation Format, 7-bit form). This encoding supports all Unicode character values, and can also be accessed as code page 65000.

UTF-8 encoding: (UTF-8 stands for UCS Transformation Format, 8-bit form). This encoding supports all Unicode character values, and can also be accessed as code page 65001.

UCS-2 encoding: Some applications (especially those that are Web based) must deal with Unicode data that is encoded with the UTF-8 encoding method. SQL Server uses a different Unicode encoding (UCS-2) and does not recognize UTF-8 as valid character data.

UTF-16 encoding: This encoding stores the basic Unicode characters using single 16 bit units and others characters using two 16 bit units. UTF-16 is the primary encoding mechanism used by Microsoft Windows Client and Server OS.

***Note**: The UCS-2 encoding scheme is actually a subset of the UTF-16 scheme. Every UCS-2 encoded code point is identical to the encoding of the same code point in UTF-16. Also, most new implementations using the Unicode standard now employ UTF-16, UTF-8 or UTF-32 instead of UCS-2.*

Encodings in SQL Server

Collation

The physical storage of character strings in Microsoft SQL Server 2008 is controlled by collations.

A collation specifies the bit patterns that represent each character and the rules by which characters are sorted and compared.

Each SQL Server collation specifies three properties:

- The sort order to use for Unicode data types (nchar and nvarchar). A sort order defines the sequence in which characters are sorted, and the way characters are evaluated in comparison operations.
- The sort order to use for non-Unicode character data types (char, varchar, and text).
- The code page used to store non-Unicode character data.

SQL Server 2008 supports objects that have different collations being stored in a single database. Separate SQL Server 2008 collations can be specified down to the level of columns. Each column in a table can be assigned different collations. Versions of SQL Server before SQL Server 2000 support only one collation for each instance of SQL Server. All databases and database objects created in an instance of SQL Server 7.0 or earlier have the same collation.

SQL Server 2008 supports the following code pages.

Code page	Description
437	MS-DOS U.S. English

850	Multilingual (MS-DOS Latin1)
874	Thai
932	Japanese
936	Chinese (Simplified)
949	Korean
950	Chinese (Traditional)
1250	Central European
1251	Cyrillic
1252	Latin1 (ANSI)
1253	Greek
1254	Turkish
1255	Hebrew
1256	Arabic
1257	Baltic
1258	Vietnamese

Microsoft SQL Server collations can be categorized in two groups: Windows collations and SQL Server collations.

Windows Collations

Windows collations are collations defined for SQL Server to support the Windows system locales available for the operating system on which SQL Server instances are installed. For information on new Windows collations support (collations based on Windows system locales) added in SQL Server 2008 and all other Windows collations, see Windows Collation Name (Transact-SQL).

By specifying a Windows collation for SQL Server, the instance of SQL Server uses the same code pages and sorting and comparison rules as an application that is running on a computer for which you have specified the associated Windows locale. For example, the French Windows collation for SQL Server matches the collation attributes of the French locale for Windows.

There are more Windows locales than there are SQL Server Windows collations. The names of Windows locales are based on language and territory; for example, French (Canada). However, several languages share common alphabets and rules for sorting and comparing characters. For example, several Windows locales, including all the Portuguese and English Windows locales, use the Latin1 code page (1252) and follow a common set of rules for sorting and comparing characters. Latin1_General, the SQL Server-supported Windows collation based on the 1252 code page and sorting rules, supports all of these Windows locales.

Also, Windows locales specify attributes that are not covered by SQL Server supported Windows collations such as currency, date, and time formats. Because countries and regions such as Great Britain and the United States have different currency, date, and time formats, they require different Windows locales. They do not require different SQL Server collations, because they have the same alphabet and rules for sorting and comparing characters. In SQL Server, Windows collations are combined with suffixes that define sorting and comparison rules based on case, accent, kana, and width sensitivity. For more information about these suffixes, see Windows Collation Sorting Styles.

These are some examples of Windows collation names:

- Latin1_General_CI_AS

Collation uses the Latin1 General dictionary sorting rules, code page 1252, is case-insensitive and accent-sensitive.

- Estonian_CS_AS

  Collation uses the Estonian dictionary sorting rules, code page 1257, is case-sensitive and accent-sensitive.

Please see the appendix for a list of Microsoft SQL Server 2008 Windows collations.

SQL Server collations

SQL Server collations are a compatibility option to match the attributes of common combinations of code-page number and sort orders that have been specified in earlier versions of SQL Server. Many of these collations support suffixes for case, accent, kana, and width sensitivity, but not always. For more information, see Using SQL Server Collations.

SQL Server collations apply non-Unicode sorting rules to non-Unicode data, and Unicode sorting rules to Unicode data, by using a corresponding Windows collation for the Unicode data. This difference can cause inconsistent results for comparisons of the same characters. Therefore, if you have a mix of Unicode and non-Unicode columns in your database, they should all be defined by using Windows collations so that the same sorting rules are used across Unicode and non-Unicode data.

To maintain compatibility with earlier versions of SQL Server, or applications that were developed with SQL Server collations in earlier versions of SQL Server, SQL Server offers the SQL_Latin1_General_CP1_CI_AS collation as the default collation for server installations on computers that use the English (United States) Windows system locale.

There can be differences in performance between Windows collations and SQL Server collations. For more information, see Storage and Performance Effects of Unicode.

Character data types

SQL Server 2008 supports two categories of character data types:

- The Unicode data types nchar and nvarchar. These data types use the Unicode character representation. Code pages do not apply to these data types.
- The non-Unicode character data types char, varchar, and text. These data types use the character representation scheme defined in a single or double-byte code page.

Collations do not control the code page used for Unicode columns, only attributes such as comparison rules and case sensitivity.

International Data and Unicode

The easiest way to manage character data in international databases is to always use the Unicode nchar and nvarchar data types in place of their non-Unicode equivalents (char, varchar, and text).

If all the applications that work with international databases also use Unicode variables instead of non-Unicode variables, character translations do not have to be performed anywhere in the system. All clients will see exactly the same characters in data as all other clients.

For systems that could use single-byte code pages, the fact that Unicode data needs twice as much storage space as non-Unicode character data is at least partially offset by eliminating the need to convert extended characters between code pages. Systems using double-byte code pages do not have this issue.

For instance, **SQL Server stores all textual system catalog data in columns having Unicode data types**. The names of database objects such as tables, views, and stored procedures are stored in Unicode columns. This allows applications to be developed using only Unicode, which avoids all issues with code page conversions.

Sort order

A sort order specifies the rules used by SQL Server to interpret, collate, compare, and present character data. For example, a sort order defines whether 'a' is less than, equal to, or greater than 'b'. A sort order defines whether the collation is case-sensitive, for example whether 'm' is equal or not equal to 'M'. It also defines if the collation is accent-sensitive, for example whether 'á' is equal or not equal to 'ä'.

SQL Server uses two sort orders with each collation, one for Unicode data and another one for the character code page.

Many SQL Server collations use the same code page, but have a different sort order for the code page.

This allows applications to choose:

- Whether characters will simply be sorted based on the numeric value represented by their bit patterns. Binary sorting is fastest because SQL Server does not have to make any adjustments and can use fast, simple sorting algorithms. Binary sort orders are always case-sensitive. Because the bit patterns in a code page may not be arranged in the same sequence as defined by the dictionary rules for a specific language, binary sorting sometimes does not sort characters in a sequence users who speak that language might expect.
- Between case-sensitive or case-insensitive behavior.
- Between accent-sensitive or accent-insensitive behavior.

Encodings in the .Net Framework

The .NET Framework is a platform for building, deploying, and running Web services and applications that provide a highly productive, standards-based, multi-language environment for integrating existing or legacy investments with next-generation applications and services.

The .NET Framework uses Unicode UTF-16 to represent characters, although in some cases it uses UTF-8 internally.

The System.Text namespace provides classes that allow you to encode and decode characters, with support that includes the following encodings:

- Unicode UTF-16 encoding. Use the UnicodeEncoding class to convert characters to and from UTF-16 encoding.
- Unicode UTF-8 encoding. Use the UTF8Encoding class to convert characters to and from UTF-8 encoding.
- ASCII encoding. ASCII encodes the Latin alphabet as single 7-bit characters. Because this encoding only supports character values from U+0000 through U+007F, in most cases it is inadequate for internationalized applications. You can use the ASCIIEncoding class to convert characters to and from ASCII encoding whenever you need to interoperate with legacy encodings and systems.
- Windows/ISO Encodings. The System.Text.Encoding class provides support for a wide range of Windows/ISO encodings.

The .NET Framework provides support for data encoded using code pages. You can use the Encoding.GetEncoding Method (Int32) to create a target encoding object for a specified code page. Specify a code page number as the Int32 parameter.

The one additional type of support introduced to ASP.NET is the ability to clearly distinguish between file, request, and response encodings. To set the encoding in ASP for code, page directives, and configuration files, you'll need to do the following.

In code:

```
Response.ContentEncoding=<value>
Request.ContentEncoding=<value>
File.ContentEncoding=<value>
```

In Page directive:

Several new directives have been added to ASP.NET. The Language attribute must now be placed within a Page directive, as shown in the following example:

```
<%@Page Language="VB" Codepage="932"%>
<%@OutputCache Duration="10 VaryByParam="location"%>
```

But for migration purposes, the shorter ASP-style syntax is also supported for the Page directive only.

```
<%@ Language="VB" Codepage="932"%>
<%@OutputCache Duration="10 VaryByParam="location"%>
```

Directives can be located anywhere in an .aspx file, but standard practice is to place them at the beginning of the file. Case is not important in ASP.NET directive statements, and quotes are not required around the attribute values.

In a configuration file:

You can set the globalization settings in the application web.config file. If not present, the settings are the ones settled in the machine.config file, and if none, the system's default ANSI code page.

Here is the syntax in the configuration file:

```
<configuration>
        <globalization
        enableClientBasedCulture="true|false"
        requestEncoding="any valid encoding string"
        responseEncoding="any valid encoding string"
        fileEncoding="any valid encoding string"

        responseHeaderEncoding = "any valid encoding string"
        resourceProviderFactoryType = string
        enableBestFitResponseEncoding = "true|false"

        culture="any valid culture string"
        uiCulture="any valid culture string"/>
</configuration>
```

For more information about setting globalization in configuration files, please go to http://msdn.microsoft.com/en-us/library/hy4kkhe0.aspx.

Encodings in Web Pages

However, Web pages currently consist of content that can be in Windows or other character-encoding schemes besides Unicode. Therefore, when form or query-string values come in from the browser in an HTTP request, they must be converted from the character set used by the browser into Unicode for processing by the .Net Framework. Similarly, when output is sent back to the browser, any strings returned by scripts must be converted from Unicode back to the code page used by the client.

Generally speaking, there are four different ways of setting the character set or the encoding of a Web page:

- **Windows code pages or ISO character encodings**: With this approach, you can select from the list of supported code pages to create your Web content. The downside of this approach is that you are limited to languages that are included in the selected character set, making true multilingual Web content impossible. This limits you to a single-script Web page.

- **Number entities:** Number entities can be used to represent a few symbols out of the currently selected code page or encoding. Let's say, for example, you have decided to create a Web page using the previous approach with the Latin ISO charset 8859-1. Now you also want to display some Greek characters in a mathematical equation; Greek characters, however, are not part of the Latin code page. Unfortunately, this approach makes it impossible to compose large amounts of text and makes editing your Web content very hard.

- **UTF-16**: Unlike Win32 applications where UTF-16 is by far the best approach, for Web content UTF-16 can be used safely only on Windows NT networks that have full Unicode support. Therefore, this is not a suggested encoding for Internet sites where the capabilities of the client Web browser as well the network Unicode support are not known.

- **UTF-8**: **This Unicode encoding is the best and safest approach for multilingual Web pages**. It allows you to encode the whole repertoire of Unicode characters. Also, all versions of Internet Explorer 4 and later support this encoding, which is not restricted to network or wire capabilities. The UTF-8 encoding allows you to create multilingual Web content without having to change the encoding based on the target language.

Setting and encoding Web Pages

Since Web content is currently based on Windows or other encoding schemes, you'll need to know how to set and manipulate encodings. The following describes how to do this for HTML pages, Active Server Pages (ASP), ASP.Net, and XML pages.

Internet Explorer uses the character set specified for a document to determine how to translate the bytes in the document into characters on the screen or on paper.

By default, Internet Explorer uses the character set specified in the HTTP content type returned by the server to determine this translation. If this parameter is not given, Internet Explorer uses the character set specified by the meta-element in the document, taking into account the user's preferences if no meta-element is specified.

To apply a character set to an entire document, you must insert the meta-element before the body element. For clarity, it should appear as the first element after the head, so that all browsers can translate the meta-element before the document is parsed. The meta-element applies to the document containing it. This means, for example, that a compound document (a document consisting of two or more documents in a set of frames) can use different character sets in different frames. Here is how it works:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=<value>">
```

You substitute with any supported character-set-friendly name (for example, UTF-8) or any code-page name (for example, windows 1251). See the appendix for more code page name.

Working with Unicode Data in Microsoft Platform

Unicode Basics

Storing data in multiple languages within one database is difficult to manage when you use only character data and code pages. It is also difficult to find one code page for the database that can store all the required language-specific characters. Additionally, it is difficult to guarantee the correct translation of special characters when being read or updated by different clients running various code pages. **Databases that support international clients should always use Unicode data types instead of non-Unicode data types**.

For example, consider a database of customers in North America that must handle three major languages:

- Spanish names and addresses for Mexico
- French names and addresses for Quebec
- English names and addresses for the rest of Canada and the United States

When you use only character columns and code pages, you must take care to make sure the database is installed with a code page that will handle the characters of all three languages. You must also take care to guarantee the correct translation of characters from one of the languages when read by clients running a code page for another language.

With the growth of the Internet, it is even more important to support many client computers that are running different locales. Selecting a code page for character data types that will support all the characters required by a worldwide audience would be difficult.

The easiest way to manage character data in international databases is to always use the Unicode nchar, nvarchar, and nvarchar(max) data types, instead of their non-Unicode equivalents, char, varchar, and text.

Unicode is a standard for mapping code points to characters. Because it is designed to cover all the characters of all the languages of the world, there is no need for different code pages to handle different sets of characters. SQL Server supports the Unicode Standard, Version 3.2.

If all the applications that work with international databases also use Unicode variables instead of non-Unicode variables, character translations do not have to be performed anywhere in the system. Clients will see the same characters in the data as all other clients.

SQL Server stores all textual system catalog data in columns having Unicode data types. The names of database objects, such as tables, views, and stored procedures, are stored in Unicode columns. This enables applications to be developed by using only Unicode, and helps avoid all issues with code page conversions.

Storage and Performance Effects of Unicode

SQL Server stores Unicode data by using the UCS-2 encoding scheme. Under this mechanism, all Unicode characters are stored by using 2 bytes.

The difference in storing character data between Unicode and non-Unicode depends on whether non-Unicode data is stored by using double-byte character sets. All non-East Asian languages and the Thai language store non-Unicode characters in single bytes. Therefore, storing these languages as Unicode uses two times the space that is used specifying a non-Unicode code page. On the other hand, the non-Unicode code pages of many other Asian languages specify character storage in double-byte character sets (DBCS). Therefore, for these languages, there is almost no difference in storage between non-Unicode and Unicode.

The following table shows the non-Unicode code pages that specify character data storage in double-byte character sets.

Language	Code page
Simplified Chinese	936
Traditional Chinese	950
Japanese	932
Korean	949

The effect of Unicode data on performance is complicated by a variety of factors that include the following:

- The difference between Unicode sorting rules and non-Unicode sorting rules
- The difference between sorting double-byte and single-byte characters
- Code page conversion between client and server

SQL Server performs string comparisons of non-Unicode data defined with a Windows collation by using Unicode sorting rules. Because these rules are much more complex than non-Unicode sorting rules, they are more resource-intensive. So, although Unicode sorting rules are frequently more expensive, there is generally little difference in performance between Unicode data and non-Unicode data defined with a Windows collation.

The only case when SQL Server uses non-Unicode sorting rules is on non-Unicode data that is defined by using SQL Server collation. Sorts and scans in this instance are generally faster than when Unicode sorting rules apply. Unicode sorting rules apply to all Unicode data, defined by using either a Windows collation or SQL Server collation.

Of secondary importance, sorting lots of Unicode data can be slower than non-Unicode data, because the data is stored in double bytes. On the other hand, sorting Asian characters in Unicode is faster than sorting Asian DBCS data in a specific code page, because DBCS data is actually a mixture of single-byte and double-byte widths, while Unicode characters are fixed-width.

Other performance issues are primarily determined by the issue of converting the encoding mechanism between an instance of SQL Server and the client. Generally, the effects on performance of client/server code-page conversion are negligible. Nevertheless, you should understand what is occurring at this layer.

The **ODBC API, version 3.6 or earlier, and the DB-Library API do not recognize Unicode**. For clients that use data access methods defined by these APIs, resources are used to implicitly convert Unicode data to the client code page. Also, **there is a risk of data corruption on the client side** when the client code page does not recognize certain Unicode characters.

Later versions of ODBC, starting with Microsoft Data Access Components version 2.7 that was included with SQL Server version 7.0, and OLE DB and ADO **are Unicode aware** and assume a UCS-2 encoding mechanism. Therefore, if the application is Unicode enabled, there are no conversion issues when you work strictly with Unicode data from an instance of SQL Server. If a client is using a Unicode-enabled API but the data storage mechanism in the instance of SQL Server is not Unicode, there are no conversion issues. However, there is a risk that any data insert or update operations will be corrupted if the code points for any character cannot be mapped to the SQL Server code page.

Unicode Best Practices

Deciding whether to store non-DBCS data as Unicode is generally determined by an awareness of the effects on storage, and about how much sorting, conversion, and possible data corruption might happen during client interactions with the data. Sorting and conversion may affect performance, depending on where it occurs. However, for most applications the effect is negligible. Databases with well-designed indexes are especially unlikely to be affected. However, data corruption will affect not only the integrity of an application and database, but also the business as a whole.

Considering this trade-off, storing character data in a specific code page may make sense if both of the following are true:

- Conserving storage space is an issue, because of hardware limitations. Or, you are performing frequent sorts of lots of data, and testing indicates that a Unicode storage mechanism severely affects performance.
- You are sure the code pages of all clients accessing this data match yours, and that this situation will not unexpectedly change.

Most of the time, the decision to store character data, even non-DBCS data, in Unicode should be based more on business needs instead of performance. In a global economy that is encouraged by rapid growth in Internet traffic, it is becoming more important than ever to support client computers that are running different locales. Additionally, it is becoming increasingly difficult to pick a single code page that supports all the characters required by a worldwide audience.

Server-side Programming with Unicode

To make a database Unicode-aware involves defining Unicode-aware client interactions in addition to using the **nchar**, **nvarchar**, and **nvarchar(max)** data types to define Unicode storage. You can define Unicode-aware client interactions by performing the following on the database server side:

- Switch from non-Unicode data types to Unicode data types in table columns and in CONVERT() and CAST() operations.
- Substitute using ASCII() and CHAR() functions with their Unicode equivalents, UNICODE() and NCHAR().
- Define variables and parameters of stored procedures and triggers in Unicode.
- Prefix Unicode character string constants with the letter N.

Using UNICODE(), NCHAR(), and other Functions

The ASCII() function returns the non-Unicode character code of the character passed in. Therefore, use the counterpart **UNICODE()** function for Unicode strings where you would use the ASCII function on non-Unicode strings. The same is true of the CHAR function; NCHAR is its Unicode counterpart.

Because the **SOUNDEX()** function is defined based on English phonetic rules, it is not meaningful on Unicode strings unless the string contains only the Latin characters A through Z and a through z.

ASCII, CHAR, and SOUNDEX can be passed Unicode parameters, but these arguments are implicitly converted to non-Unicode strings. This could cause the possible loss of Unicode characters before processing, because these functions operate on non-Unicode strings by definition.

Besides the UNICODE() and NCHAR() functions, **the following string manipulation functions support Unicode wherever possible**:

- CHARINDEX()
- LEFT()
- LEN()
- UPPER()
- LOWER()
- LTRIM()
- RTRIM()
- PATINDEX()
- REPLACE()
- QUOTENAME()
- REPLICATE()
- REVERSE()
- STUFF()
- SUBSTRING()
- UNICODE()

These functions accept Unicode arguments, respect the 2-byte character boundaries of Unicode strings, and use Unicode sorting rules for string comparisons when the input parameters are Unicode.

Defining Parameters in stored Procedures
Defining parameters with a Unicode data type guarantees that client requests or input are implicitly converted to Unicode on the server and not corrupted in the process. If the parameter is specified as an OUTPUT parameter, a Unicode type also minimizes the chance of corruption on its way back to the client.

In the following stored procedure, the variable is declared as a Unicode data type.

```
CREATE PROCEDURE Product_Info
 @name nvarchar(40)
AS
SELECT p.ListPrice, v.Name
 FROM Production.Product p
 INNER JOIN Purchasing.ProductVendor pv
 ON p.ProductID = pv.ProductID
 INNER JOIN Purchasing.Vendor v
 ON pv.VendorID = v.VendorID
WHERE p.Name = @name;
```

Using the N Prefix

Unicode string constants that appear in code executed on the server, such as in stored procedures and triggers, must be preceded by the capital letter N. This is true even if the column being referenced is already defined as Unicode. Without the N prefix, the string is converted to the default code page of the database. This may not recognize certain characters.

For example, the stored procedure created in the previous example can be executed on the server in the following way:

```
EXECUTE Product_Info @name = N'Chain'
```

The requirement to use the N prefix applies to both string constants that originate on the server and those sent from the client.

Client-side Programming with Unicode

The topics in this section explain how to preserve the integrity of character data when you program client-side database applications.

Managing Data Conversion between Client/Server Code Pages

This topic describes how to preserve the integrity of character data when the database does not store the character data by using Unicode data types and when your client-side applications that interact with the data are also not Unicode-aware. In this situation, the code page of your data storage and the code page of the client-side application should be the same. If these code pages differ, the conversion that occurs between the client and the server might cause the loss of some characters.

Disabling the AutoTranslate feature of the SQL Server ODBC driver to insert data defined by a different code page from the server is not supported. Also, even if AutoTranslate is disabled, it does not prevent code page translation for SQL language events. The result is that if the client and database code pages do not match, code page translation will generally be applied to any non-Unicode character string that is sent to or from the server.

If you can, you should avoid this situation. **The best choice for a code page-specific server is to communicate only with the clients using the same code page**. The second-best choice is to use another code page that has almost the same character set. For example, code page 1252 (Latin1) and code page 850 (Multilingual Latin1) can store almost the same set of characters, so that most characters in these two code pages can be converted from one code page to another without data loss.

If you must communicate with clients using different code pages, the supported solution is to store your data in Unicode columns. If any one of these options is not feasible, the other alternative is to store the data in binary columns using the binary, varbinary, or varbinary(max) data types. However, binary data can only be sorted and compared in binary order. This makes it less flexible than character data.

Managing Data Conversion between a Unicode Server and a Non-Unicode Client

This topic describes how to preserve the integrity of character data when the server-side data storage is in Unicode, but the client-side application that interacts with the data uses a specific code page.

Data Input

When non-Unicode data is sent from the client to be stored on the server in Unicode, data from any client with any code page can be stored correctly if one of the following conditions is true:

- Character strings are sent to the server as parameters of a remote procedure call (RPC).
- String constants are preceded with the capital letter N. This is required regardless of whether your client-side application is Unicode-aware. Without the N prefix, SQL Server will convert the string to the code page that corresponds to the default collation of the database. Any characters not found in this code page will be lost.

Data Retrieval

If the client application is not Unicode-enabled and retrieves the data into non-Unicode buffers, a client will only be able to retrieve or modify data that can be represented by the client machine's code page. This means that ASCII characters can always be retrieved, because the representation of ASCII characters is the same in all code pages, while any non-ASCII data depends on code-page-to-code-page conversion.

For example, suppose you have an application that is currently running only in the United States (U.S.), but is deployed to Japan. Because the SQL Server database is Unicode-aware, both the English and Japanese text can be stored in the same tables, even though the application has not yet been modified to deal with text as Unicode. As long as the application complies with one of the two previous options, Japanese users can use the non-Unicode application to input and retrieve Japanese data, and U.S. users can input and retrieve English data. All data from both sets of users is stored intact in the same column of the database and represented as Unicode. In this situation, a Unicode-enabled reporting application that generates reports that span the complete data set can be

deployed. However, English users cannot view the Japanese rows, because the application cannot display any characters that do not exist in their code page (1252).

This situation might be acceptable if the two groups of users do not have to view each other's records. If an application user must be able to view or modify records with text that cannot be represented by a single code page, there is no alternative but to modify the application so that it can use Unicode.

Web-based Applications

If the client-side program is Web-based or connects to an Active Server Pages (ASP) page, there are metadata specifications on both the client-side HTML page and the server-side ASP page. These specifications must be made to specify how character strings should be converted between the server, the ASP engine, and the client browser.

On the client side HTML page, the META attribute must specify that the character set data should be converted to the encoding scheme of the client by specifying a CHARSET code. For example, the following HTML page instructs the client to convert character data to the 950 (Chinese Traditional) code page by specifying big5 as the CHARSET code.

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; CHARSET=big5">
<!--


-->
</HEAD>
```

```
<BODY>
<!--
 body
-->
</BODY>
</HTML>
```

On the server-side ASP page, you must instruct the ASP Web application what code page the client browser is using. You can specify the Session.CodePage property, or the @CodePage directive. These methods will handle the conversion of data from server to client and also both GET and POST client requests. In the following examples, both methods are used to specify conversion to and from the code page of the client, which is 950 (Chinese Traditional).

```
<%@ Language=VBScript codepage=950 %>
<% Session.CodePage=950 %>
```

And finally, you must remember to prefix any string literals with the letter N.

Managing Data Conversion between Unicode Encoding Schemes

This topic describes how to preserve the integrity of character data when both server-side data storage and the client application that interacts with the data are Unicode-enabled, but use different Unicode encoding schemes. SQL Server stores Unicode in the UCS-2 encoding scheme. However, many clients process Unicode in another encoding scheme, generally UTF-8. This scenario frequently occurs for Web-based applications.

Because you are essentially still converting from one encoding scheme to another; many of the same solutions discussed in the previous topics also apply. Unicode character string constants sent to the server must be preceded with a capital N. For Web-based applications, you specify the CHARSET code under the META attribute of the client-side HTML page. For example, specify CHARSET = utf-8 if the Unicode encoding scheme is UTF-8. On the server side, specify the encoding scheme of the client by using the Session.CodePage property or the @Codepage directive. For example, codepage=65001 specifies a UTF-8 encoding scheme. If you follow these directions, Internet Information Services (IIS) 5.0 or later versions will seamlessly handle the conversion from UTF-8 to UCS-2 and back without additional effort on your part.

In Visual Basic applications, character strings are processed in the UCS-2 encoding scheme. Therefore, you do not have to specify encoding scheme conversion explicitly between these applications and an instance of SQL Server.

Using Unicode with bcp and OPENROWSET

To prevent character loss when you use the bcp utility to copy data between servers that have different code pages, you can specify that the data be copied in Unicode format. For more information, see Copying Data Between Different Collations.

Unicode format can also be specified when you use OPENROWSET to access external data. For more information, see OPENROWSET (Transact-SQL).

Using Unicode with XML Data

SQL Server stores XML data using the UTF-16 encoding scheme. Because UTF-16 data is variable-width, it is processed according to a byte-oriented protocol. This means that UTF-16 data can be treated in a way that is independent of the byte ordering on different computers (little endian versus big endian). Therefore, UTF-16 is well-suited for traversing different computers that use different encodings and byte-ordering systems. Because XML data is typically shared widely across networks, it makes sense to maintain the default UTF-16 storage of XML data in your database, and when you export XML data to clients.

If you must specify a different encoding, you can use FOR XML requests and specify the following:

- The Output Encoding property of an XML-formatted data stream Response object in Active Server Pages (ASP).
- For example, the following ASP code tells the browser to display an incoming XML data stream in UCS-2:

```
<% cmdXML.Properties("Output Encoding") = "UCS-2" %>
```

- An output encoding in a URL when you make an HTTP request.
- The following example specifies UCS-2 as the output encoding of the XML document returned by this request:

```
http://IISServer/nwind?sql=SELECT+*+FROM+Customers+FOR+XML+AUTO&outputencoding=UCS-2
```

- An output encoding in an XML template or style sheet.
- The following example specifies UCS-2 as the output encoding in the header of this XML template document:

```
<?xml version ='1.0' encoding='UCS-2'?>
 <root xmlns:sql='urn:schemas-microsoft-com:xml-sql'
 sql:xsl='MyXSL.xsl'>
 <sql:query>
```

```
SELECT FirstName, LastName FROM Employees FOR XML AUTO

 </sql:query>

</root>
```

SQL Server Globalization Development Recommendations

Use nchar/nvarchar instead of char/varchartype

Nchar/nvarchar type is the Unicode data type used by SQL Server, which can store any characters defined by Unicode.org. Char/varchar type is always associated with a code page, so the number of supported characters is limited. For example:

- Starting from Windows 2000, Windows has full Unicode support implemented by using UTF-16 Encoding. Windows APIs take WCHAR* as input which represent a Unicode String.
- Virtual C++ has WCHAR, which is Unicode char type.
- .Net String is Unicode String only encoded in UTF-16.
- Java String is Unicode String only encoded in UTF-16.
- Char type might have a data corruption issue if your client locale is different from the server locale.

Deprecate old code page technique

Use Unicode in your application and in your SQL Server database. The only benefit of using char type is the space saving for single byte code page. You can always use the Data Compression feature in SQL Server 2008 if you care about disk space, which can save more space than using char type.

Use N for your string literal in T-SQL

Remember to put N' for your string literal in T-SQL. String literal N'abc' is a Unicode nvarcharstring literal. Literal 'ab©' is a varchar literal, it always associates with a code page (string literal always use current database's collation).Suppose char © is not in the code page, you will get a question mark (?) when inserting the literal into a table, even the column is nvarchar type.

Use nvarchar instead of nchar

Use nvarchar instead of nchar. The difference between nvarchar and nchar is the storage. A column with nchar(30) type always take 60 bytes to store on the disk, even the value is a single character. The data size for an nvarchar(30) type column is not fixed; it varies row by row or value by value. A single character value takes 2 bytes to store, and a value with 30 characters long takes 60bytes to be stored. Another different between nvarchar and nchar type is the performance. Nchar types always stored in fixed location in every row, which can be retrieved faster than nvarchar type which is stored in different location for different row. However, I believe the benefit of less stored space for nvarchar types usually overcomes the cost of locating the value in a row.

Avoid column collation that is different from the dataset collation

If possible, avoid using a column collation which is different with the database's collation. You will have less collation conflict. If you want a query use a special collation's sorting rule, use explicit collate clause in that query.

Use Windows Collation instead of SQL Server Collation

Use Windows collation instead of SQL Server collation. The only exception here is the default collation for en_US locale which issql_latin1_general_cp1_ci_as.

Never store UTF-8 string in varchar types

Never store UTF-8 string in varchar types, you *will* get data corruption.

String comparison and trailing spaces

String comparison always ignores trailing spaces (Unicode U+0020), no matter what collation you are using.

LEN Function reminder

LEN function always returns the number of characters excluding the trailing spaces. DataLength function returns the storage size in term of bytes.

_BIN2 Collation

Use _BIN2 collation, instead of _BIN collation if you want binary, code page base string comparison behavior.

Working with other data sources in Microsoft Platform

The following section deals with inserting data from source systems into Microsoft environment. Each part describes the possibilities and constraints of the specified method.

Working with UTF-16 source data in SQL Server (Unicode)

SQL Server offers the following possibilities when working with UTF-16 data sets:

- SQL Server can store any Unicode characters in UTF-16 encoding. The reason is that the storage format for UCS-2 and UTF-16 are the same.
- SQL Server can display any Unicode characters in UTF-16 encoding. The reason is that we internally call Windows functions to display characters, the Windows functions and fonts can take care of the supplementary character (a character take 4 bytes in UTF-16) correctly.
- SQL Server can input any Unicode characters in UTF-16 encoding. The reason is that we internally call Windows IMEs (Input Method Editors) to input, the Windows IMEs can take care of the supplementary character (a character take 4 bytes in UTF-16) correctly.
- SQL Server can sort/compare any defined Unicode characters in UTF-16 encoding. Note, not all code points are mapped to valid Unicode characters. For example, The Unicode Standard, Version 5.1 defines code points for around 10,000 characters. All these characters can be compared/sorted in SQL Server latest version: SQL Server 2008.

SQL Server has the following constraints when working with UTF-16 data sets:

SQL Server **cannot detect invalid UTF-16 sequences**. Unpaired surrogate character is not valid in UTF-16 encoding, but SQL Server accept it as valid. Note, in reality, it is unlikely end-user will input invalid UTF-16 sequence since they are not supported in any language or by any IMEs.

- SQL Server **treats a UTF-16 supplementary character as two characters**. The Len function return 2 instead of 1 for such input.
- SQL Server **has a potential risk of breaking a UTF-16 supplementary character into an un-paired surrogate character**, as in calling the substring function. Note, in the real scenario, the chance of this to happen is much lower, because supplementary characters are rare and string function will only break when it happens at the boundary. For example, calling substring(s,5,1) will break if and only if the character at index 5 is a supplementary character.

Working with UTF-8 source data in SQL Server (Unicode)

SQL Server doesn't support UTF-8 encoding for Unicode data, it supports the UTF-16 encoding. Conversion from UTF-8 to UTF-16 must be addressed through ETL.

With SQL Server Integration Services 2008, a simple Data Conversion component inserted in a package can convert data from UTF-8 to UTF-16.

Working with Non-Unicode source data in SQL Server (Unicode)

Expected inputs

Code page of the source must be compatible with the code pages supported by ETL tools.

Illustration: Informatica mechanism
Code page compatibility matrix

With the "Codepage Relaxation" feature, Src - Informatica Server - Tgt do not need to be in compatible codepages.

However there are some limitations:

- Informatica Server should run in UNICODE mode.
- User has to make sure the code page of Src should be compatible with that of Tgt.
- Both client and server have to set code page relaxation on.

Illustration: SQL Server Integration Services mechanism

With SQL Server 2008 Integration Services, a simple Data Conversion component inserted in a package can convert data from a source code page to Unicode. A prerequisite to this step is to make sure that SQL Server 2008 Integration Services is compatible with the source code page.

For a list of compatible code pages, please refer to Appendix 3.

Working with source date in SQL Server (Non-Unicode)

Code page of the source must be compatible with the code page of SQL Server if you want to store data in Non-Unicode data types.

SQL Server 2008 supports the following code pages.

Code page	Description
1258	Vietnamese
1257	Baltic
1256	Arabic
1255	Hebrew
1254	Turkish
1253	Greek
1252	Latin1 (ANSI)
1251	Cyrillic
1250	Central European
950	Chinese (Traditional)
949	Korean
936	Chinese (Simplified)
932	Japanese
874	Thai
850	Multilingual (MS-DOS Latin1)
437	MS-DOS U.S. English

Appendix

Common code page list

Here is the list of the most popular code pages:

Code Page	Name	DisplayName
37	IBM037	IBM EBCDIC (US-Canada)

437	IBM437	OEM United States
500	IBM500	IBM EBCDIC (International)
708	ASMO-708	Arabic (ASMO 708)
720	DOS-720	Arabic (DOS)
737	ibm737	Greek (DOS)
775	ibm775	Baltic (DOS)
850	ibm850	Western European (DOS)
852	ibm852	Central European (DOS)
855	IBM855	OEM Cyrillic
857	ibm857	Turkish (DOS)
858	IBM00858	OEM Multilingual Latin I
860	IBM860	Portuguese (DOS)
861	ibm861	Icelandic (DOS)
862	DOS-862	Hebrew (DOS)
863	IBM863	French Canadian (DOS)
864	IBM864	Arabic (864)
865	IBM865	Nordic (DOS)

866	cp866	Cyrillic (DOS)
869	ibm869	Greek, Modern (DOS)
870	IBM870	IBM EBCDIC (Multilingual Latin-2)
874	windows-874	Thai (Windows)
875	cp875	IBM EBCDIC (Greek Modern)
932	shift_jis	Japanese (Shift-JIS)
936	gb2312	Chinese Simplified (GB2312)
949	ks_c_5601-1987	Korean
950	big5	Chinese Traditional (Big5)
1026	IBM1026	IBM EBCDIC (Turkish Latin-5)
1047	IBM01047	IBM Latin-1
1140	IBM01140	IBM EBCDIC (US-Canada-Euro)
1141	IBM01141	IBM EBCDIC (Germany-Euro)
1142	IBM01142	IBM EBCDIC (Denmark-Norway-Euro)
1143	IBM01143	IBM EBCDIC (Finland-Sweden-Euro)
1144	IBM01144	IBM EBCDIC (Italy-Euro)
1145	IBM01145	IBM EBCDIC (Spain-Euro)

1146	IBM01146	IBM EBCDIC (UK-Euro)
1147	IBM01147	IBM EBCDIC (France-Euro)
1148	IBM01148	IBM EBCDIC (International-Euro)
1149	IBM01149	IBM EBCDIC (Icelandic-Euro)
1200	utf-16	Unicode
1201	unicodeFFFE	Unicode (Big-Endian)
1250	windows-1250	Central European (Windows)
1251	windows-1251	Cyrillic (Windows)
1252	Windows-1252	Western European (Windows)
1253	windows-1253	Greek (Windows)
1254	windows-1254	Turkish (Windows)
1255	windows-1255	Hebrew (Windows)
1256	windows-1256	Arabic (Windows)
1257	windows-1257	Baltic (Windows)
1258	windows-1258	Vietnamese (Windows)
1361	Johab	Korean (Johab)
10000	macintosh	Western European (Mac)

10001	x-mac-japanese	Japanese (Mac)
10002	x-mac-chinesetrad	Chinese Traditional (Mac)
10003	x-mac-korean	Korean (Mac)
10004	x-mac-arabic	Arabic (Mac)
10005	x-mac-hebrew	Hebrew (Mac)
10006	x-mac-greek	Greek (Mac)
10007	x-mac-cyrillic	Cyrillic (Mac)
10008	x-mac-chinesesimp	Chinese Simplified (Mac)
10010	x-mac-romanian	Romanian (Mac)
10017	x-mac-ukrainian	Ukrainian (Mac)
10021	x-mac-thai	Thai (Mac)
10029	x-mac-ce	Central European (Mac)
10079	x-mac-icelandic	Icelandic (Mac)
10081	x-mac-turkish	Turkish (Mac)
10082	x-mac-croatian	Croatian (Mac)
20000	x-Chinese-CNS	Chinese Traditional (CNS)
20001	x-cp20001	TCA Taiwan

20002	x-Chinese-Eten	Chinese Traditional (Eten)
20003	x-cp20003	IBM5550 Taiwan
20004	x-cp20004	TeleText Taiwan
20005	x-cp20005	Wang Taiwan
20105	x-IA5	Western European (IA5)
20106	x-IA5-German	German (IA5)
20107	x-IA5-Swedish	Swedish (IA5)
20108	x-IA5-Norwegian	Norwegian (IA5)
20127	us-ascii	US-ASCII
20261	x-cp20261	T.61
20269	x-cp20269	ISO-6937
20273	IBM273	IBM EBCDIC (Germany)
20277	IBM277	IBM EBCDIC (Denmark-Norway)
20278	IBM278	IBM EBCDIC (Finland-Sweden)
20280	IBM280	IBM EBCDIC (Italy)
20284	IBM284	IBM EBCDIC (Spain)
20285	IBM285	IBM EBCDIC (UK)

20290	IBM290	IBM EBCDIC (Japanese katakana)
20297	IBM297	IBM EBCDIC (France)
20420	IBM420	IBM EBCDIC (Arabic)
20423	IBM423	IBM EBCDIC (Greek)
20424	IBM424	IBM EBCDIC (Hebrew)
20833	x-EBCDIC-KoreanExtended	IBM EBCDIC (Korean Extended)
20838	IBM-Thai	IBM EBCDIC (Thai)
20866	koi8-r	Cyrillic (KOI8-R)
20871	IBM871	IBM EBCDIC (Icelandic)
20880	IBM880	IBM EBCDIC (Cyrillic Russian)
20905	IBM905	IBM EBCDIC (Turkish)
20924	IBM00924	IBM Latin-1
20932	EUC-JP	Japanese (JIS 0208-1990 and 0212-1990)
20936	x-cp20936	Chinese Simplified (GB2312-80)
20949	x-cp20949	Korean Wansung
21025	cp1025	IBM EBCDIC (Cyrillic Serbian-Bulgarian)
21866	koi8-u	Cyrillic (KOI8-U)

28591	iso-8859-1	Western European (ISO)
28592	iso-8859-2	Central European (ISO)
28593	iso-8859-3	Latin 3 (ISO)
28594	iso-8859-4	Baltic (ISO)
28595	iso-8859-5	Cyrillic (ISO)
28596	iso-8859-6	Arabic (ISO)
28597	iso-8859-7	Greek (ISO)
28598	iso-8859-8	Hebrew (ISO-Visual)
28599	iso-8859-9	Turkish (ISO)
28603	iso-8859-13	Estonian (ISO)
28605	iso-8859-15	Latin 9 (ISO)
29001	x-Europa	Europa
38598	iso-8859-8-i	Hebrew (ISO-Logical)
50220	iso-2022-jp	Japanese (JIS)
50221	csISO2022JP	Japanese (JIS-Allow 1 byte Kana)
50222	iso-2022-jp	Japanese (JIS-Allow 1 byte Kana - SO/SI)
50225	iso-2022-kr	Korean (ISO)

50227	x-cp50227	Chinese Simplified (ISO-2022)
51932	euc-jp	Japanese (EUC)
51936	EUC-CN	Chinese Simplified (EUC)
51949	euc-kr	Korean (EUC)
52936	hz-gb-2312	Chinese Simplified (HZ)
54936	GB18030	Chinese Simplified (GB18030)
57002	x-iscii-de	ISCII Devanagari
57003	x-iscii-be	ISCII Bengali
57004	x-iscii-ta	ISCII Tamil
57005	x-iscii-te	ISCII Telugu
57006	x-iscii-as	ISCII Assamese
57007	x-iscii-or	ISCII Oriya
57008	x-iscii-ka	ISCII Kannada
57009	x-iscii-ma	ISCII Malayalam
57010	x-iscii-gu	ISCII Gujarati
57011	x-iscii-pa	ISCII Punjabi
65000	utf-7	Unicode (UTF-7)

65001	utf-8	Unicode (UTF-8)
65005	utf-32	Unicode (UTF-32)
65006	utf-32BE	Unicode (UTF-32 Big-Endian)

Windows Collations

The following is the list of supported Windows collations in SQL Server 2008.

Windows locale	SQL Server 2008 (100)	SQL Server 2005 (90) or SQL Server 2000
Afrikaans (South Africa)	Latin1_General_100_	Latin1_General_

Albanian (Albania)	Albanian_100_	Albanian_
Alsatian (France)	Latin1_General_100_	New to SQL Server 2008
Amharic (Ethiopia)	Latin1_General_100_	New to SQL Server 2008
Arabic (Algeria)	Arabic_100_	Arabic_
Arabic (Bahrain)	Arabic_100_	Arabic_
Arabic (Egypt)	Arabic_100_	Arabic_
Arabic (Iraq)	Arabic_100_	Arabic_
Arabic (Jordan)	Arabic_100_	Arabic_
Arabic (Kuwait)	Arabic_100_	Arabic_
Arabic (Lebanon)	Arabic_100_	Arabic_
Arabic (Libya)	Arabic_100_	Arabic_
Arabic (Morocco)	Arabic_100_	Arabic_
Arabic (Oman)	Arabic_100_	Arabic_
Arabic (Qatar)	Arabic_100_	Arabic_
Arabic (Saudi Arabia)	Arabic_100_	Arabic_
Arabic (Syria)	Arabic_100_	Arabic_
Arabic (Tunisia)	Arabic_100_	Arabic_

Arabic (U.A.E.)	Arabic_100_	Arabic_
Arabic (Yemen)	Arabic_100_	Arabic_
Armenian (Armenia)	Cyrillic_General_100_	New to SQL Server 2008
Assamese (India)	Assamese_100_ [1]	New to SQL Server 2008
Azeri (Azerbaijan, Cyrillic)	Azeri_Cyrillic_100_	Azeri_Cyrillic_90_
Azeri (Azerbaijan, Latin)	Azeri_Latin_100_	Azeri_Latin_90_
Bashkir (Russia)	Bashkir_100_	New to SQL Server 2008
Basque (Basque)	Latin1_General_100_	New to SQL Server 2008
Belarusian (Belarus)	Cyrillic_General_100_	Cyrillic_General_
Bengali (Bangladesh)	Bengali_100_ [1]	New to SQL Server 2008
Bengali (India)	Bengali_100_ [1]	New to SQL Server 2008
Bosnian (Bosnia and Herzegovina, Cyrillic)	Bosnian_Cyrillic_100_	New to SQL Server 2008
Bosnian (Bosnia and Herzegovina, Latin)	Bosnian_Latin_100_	New to SQL Server 2008
Breton (France)	Breton_100_	New to SQL Server 2008
Bulgarian (Bulgaria)	Cyrillic_General_100_	Cyrillic_General_
Catalan (Catalan)	Latin1_General_100_	Latin1_General_

Chinese (Hong Kong SAR, PRC)	Chinese_Traditional_Stroke_Count_100_	Chinese_Hong_Kong_Stroke_90_
Chinese (Macao SAR)	Chinese_Traditional_Pinyin_100_	New to SQL Server 2008
Chinese (Macau)	Chinese_Traditional_Stroke_Order_100_	New to SQL Server 2008
Chinese (PRC)	Chinese_Simplified_Pinyin_100_	Chinese_PRC_90_, Chinese_PRC_
Chinese (PRC)	Chinese_Simplified_Stroke_Order_100_	Chinese_PRC_Stroke_90_, Chinese_PRC_Stroke_
Chinese (Singapore)	Chinese_Simplified_Pinyin_100_	Chinese_PRC_90_, Chinese_PRC_
Chinese (Singapore)	Chinese_Simplified_Stroke_Order_100_	New to SQL Server 2008
Chinese (Taiwan)	Chinese_Traditional_Bopomofo_100_	Chinese_Taiwan_Bopomofo_90_, Chinese_Taiwan_Bopomofo_
Chinese (Taiwan)	Chinese_Traditional_Stroke_Count_100_	Chinese_Taiwan_Stroke_90_, Chinese_Taiwan_Stroke_
Corsican (France)	Corsican_100_	New to SQL Server 2008
Croatian (Bosnia and Herzegovina, Latin)	Croatian_100_	New to SQL Server 2008
Croatian (Croatia)	Croatian_100_	Croatian_

Czech (Czech Republic)	Czech_100_	Czech_
Danish (Denmark)	Danish_Greenlandic_100_	Danish_Norwegian_
Dari (Afghanistan)	Dari_100_	New to SQL Server 2008
Divehi (Maldives)	Divehi_100_[1]	Divehi_90_
Dutch (Belgium)	Latin1_General_100_	Latin1_General_
Dutch (Netherlands)	Latin1_General_100_	Latin1_General_
English (Australia)	Latin1_General_100_	Latin1_General_
English (Belize)	Latin1_General_100_	Latin1_General_
English (Canada)	Latin1_General_100_	Latin1_General_
English (Caribbean)	Latin1_General_100_	Latin1_General_
English (India)	Latin1_General_100_	New to SQL Server 2008
English (Ireland)	Latin1_General_100_	Latin1_General_
English (Jamaica)	Latin1_General_100_	Latin1_General_
English (Malaysia)	Latin1_General_100_	New to SQL Server 2008
English (New Zealand)	Latin1_General_100_	Latin1_General_
English (Philippines)	Latin1_General_100_	Latin1_General_
English (Singapore)	Latin1_General_100_	New to SQL Server 2008

English (South Africa)	Latin1_General_100_	Latin1_General_
English (Trinidad and Tobago)	Latin1_General_100_	Latin1_General_
English (United Kingdom)	Latin1_General_100_	Latin1_General_
English (United States)	SQL_Latin1_General_CP1_	SQL_Latin1_General_CP1_
English (Zimbabwe)	Latin1_General_100_	Latin1_General_
Estonian (Estonia)	Estonian_100_	Estonian_
Faroese (Faroe Islands)	Latin1_General_100_	Latin1_General_
Filipino (Philippines)	Latin1_General_100_	New to SQL Server 2008
Finnish (Finland)	Finnish_Swedish_100_	Finnish_Swedish_
French (Belgium)	French_100_	French_
French (Canada)	French_100_	French_
French (France)	French_100_	French_
French (Luxembourg)	French_100_	French_
French (Monaco)	French_100_	French_
French (Switzerland)	French_100_	French_
Frisian (Netherlands)	Frisian_100_	New to SQL Server 2008

Galician (Spain)	Latin1_General_100_	Latin1_General_
Georgian (Georgia)	Georgian_Modern_Sort_100_	Georgian_Modern_Sort_
Georgian (Georgia)	Cyrillic_General_100_	New to SQL Server 2008
German - Phone Book Sort (DIN)	German_PhoneBook_100_	German_PhoneBook_
German (Austria)	Latin1_General_100_	Latin1_General_
German (Germany)	Latin1_General_100_	Latin1_General_
German (Liechtenstein)	Latin1_General_100_	Latin1_General_
German (Luxembourg)	Latin1_General_100_	Latin1_General_
German (Switzerland)	Latin1_General_100_	Latin1_General_
Greek (Greece)	Greek_100_	Greek_
Greenlandic (Greenland)	Danish_Greenlandic_100_	New to SQL Server 2008
Gujarati (India)	Indic_General_100_[1]	Indic_General_90_
Hausa (Nigeria, Latin)	Latin1_General_100_	New to SQL Server 2008
Hebrew (Israel)	Hebrew_100_	Hebrew_
Hindi (India)	Indic_General_100_[1]	Indic_General_90_
Hungarian (Hungary)	Hungarian_100_	Hungarian_

Hungarian Technical Sort	Hungarian_Technical_100_	Hungarian_Technical_
Icelandic (Iceland)	Icelandic_100_	Icelandic_
Igbo (Nigeria)	Latin1_General_100_	New to SQL Server 2008
Indonesian (Indonesia)	Latin1_General_100_	Latin1_General_
Inuktitut (Canada, Latin)	Latin1_General_100_	New to SQL Server 2008
Inuktitut (Syllabics) Canada	Latin1_General_100_	New to SQL Server 2008
Irish (Ireland)	Latin1_General_100_	New to SQL Server 2008
Italian (Italy)	Latin1_General_100_	Latin1_General_
Italian (Switzerland)	Latin1_General_100_	Latin1_General_
Japanese (Japan XJIS)	Japanese_XJIS_100_	Japanese_90_, Japanese_
Japanese (Japan)	Japanese_Bushu_Kakusu_100_	New to SQL Server 2008
Kannada (India)	Indic_General_100_[1]	Indic_General_90_
Kazakh (Kazakhstan)	Kazakh_100_	Kazakh_90_
Khmer (Cambodia)	Khmer_100_[1]	New to SQL Server 2008
K'iche (Guatemala)	Modern_Spanish_100_	New to SQL Server 2008
Kinyarwanda (Rwanda)	Latin1_General_100_	New to SQL Server 2008

Konkani (India)	Indic_General_100_[1]	Indic_General_90_
Korean (Korea Dictionary Sort)	Korean_100_	Korean_90_, Korean_Wansung_
Kyrgyz (Kyrgyzstan)	Cyrillic_General_100_	Cyrillic_General_
Lao (Lao PDR)	Lao_100_[1]	New to SQL Server 2008
Latvian (Latvia)	Latvian_100_	Latvian_
Lithuanian (Lithuania)	Lithuanian_100_	Lithuanian_
Lower Sorbian (Germany)	Latin1_General_100_	New to SQL Server 2008
Luxembourgish (Luxembourg)	Latin1_General_100_	New to SQL Server 2008
Macedonian (Macedonia, FYROM)	Macedonian_FYROM_100_	Macedonian_FYROM_90_
Malay (Brunei Darussalam)	Latin1_General_100_	Latin1_General_
Malay (Malaysia)	Latin1_General_100_	Latin1_General_
Malayalam (India)	Indic_General_100_[1]	New to SQL Server 2008
Maltese (Malta)	Maltese_100_	New to SQL Server 2008
Maori (New Zealand)	Maori_100_	New to SQL Server 2008

Mapudungun (Chile)	Mapudungan_100_	New to SQL Server 2008
Marathi (India)	Indic_General_100_[1]	Indic_General_90_
Mohawk (Canada)	Mohawk_100_	New to SQL Server 2008
Mongolian (Mongolia)	Cyrillic_General_100_	Cyrillic_General_
Mongolian (PRC)	Cyrillic_General_100_	New to SQL Server 2008
Nepali (Nepal)	Nepali_100_[1]	New to SQL Server 2008
Norwegian (Bokmål, Norway)	Norwegian_100_	New to SQL Server 2008
Norwegian (Nynorsk, Norway)	Norwegian_100_	New to SQL Server 2008
Occitan (France)	French_100_	New to SQL Server 2008
Oriya (India)	Indic_General_100_[1]	New to SQL Server 2008
Pashto (Afghanistan)	Pashto_100_[1]	New to SQL Server 2008
Persian (Iran)	Persian_100_	New to SQL Server 2008
Polish (Poland)	Polish_100_	Polish_
Portuguese (Brazil)	Latin1_General_100_	Latin1_General_
Portuguese (Portugal)	Latin1_General_100_	Latin1_General_
Punjabi (India)	Indic_General_100_[1]	Indic_General_90_

Quechua (Bolivia)	Latin1_General_100_	New to SQL Server 2008
Quechua (Ecuador)	Latin1_General_100_	New to SQL Server 2008
Quechua (Peru)	Latin1_General_100_	New to SQL Server 2008
Romanian (Romania)	Romanian_100_	Romanian_
Romansh (Switzerland)	Romansh_100_	New to SQL Server 2008
Russian (Russia)	Cyrillic_General_100_	Cyrillic_General_
Sami (Inari, Finland)	Sami_Sweden_Finland_100_	New to SQL Server 2008
Sami (Lule,Norway)	Sami_Norway_100_	New to SQL Server 2008
Sami (Lule, Sweden)	Sami_Sweden_Finland_100_	New to SQL Server 2008
Sami (Northern, Finland)	Sami_Sweden_Finland_100_	New to SQL Server 2008
Sami (Northern,Norway)	Sami_Norway_100_	New to SQL Server 2008
Sami (Northern, Sweden)	Sami_Sweden_Finland_100_	New to SQL Server 2008
Sami (Skolt, Finland)	Sami_Sweden_Finland_100_	New to SQL Server 2008
Sami (Southern, Norway)	Sami_Norway_100_	New to SQL Server 2008
Sami (Southern, Sweden)	Sami_Sweden_Finland_100_	New to SQL Server 2008
Sanskrit (India)	Indic_General_100_[1]	Indic_General_90_
Serbian (Bosnia and	Serbian_Cyrillic_100_	New to SQL Server 2008

Herzegovina, Cyrillic)		
Serbian (Bosnia and Herzegovina, Latin)	Serbian_Latin_100_	New to SQL Server 2008
Serbian (Serbia, Cyrillic)	Serbian_Cyrillic_100_	New to SQL Server 2008
Serbian (Serbia, Latin)	Serbian_Latin_100_	New to SQL Server 2008
Sesotho sa Leboa/Northern Sotho (South Africa)	Latin1_General_100_	New to SQL Server 2008
Setswana/Tswana (South Africa)	Latin1_General_100_	New to SQL Server 2008
Sinhala (Sri Lanka)	Indic_General_100_[1]	New to SQL Server 2008
Slovak (Slovakia)	Slovak_100_	Slovak_
Slovenian (Slovenia)	Slovenian_100_	Slovenian_
Spanish (Argentina)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Bolivia)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Chile)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Colombia)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Costa Rica)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Dominican Republic)	Modern_Spanish_100_	Modern_Spanish_

Spanish (Ecuador)	Modern_Spanish_100_	Modern_Spanish_
Spanish (El Salvador)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Guatemala)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Honduras)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Mexico)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Nicaragua)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Panama)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Paraguay)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Peru)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Puerto Rico)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Spain)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Spain, Traditional Sort)	Traditional_Spanish_100_	Traditional_Spanish_
Spanish (United States)	Latin1_General_100_	New to SQL Server 2008
Spanish (Uruguay)	Modern_Spanish_100_	Modern_Spanish_
Spanish (Venezuela)	Modern_Spanish_100_	Modern_Spanish_
Swahili (Kenya)	Latin1_General_100_	New to SQL Server 2008
Swedish (Finland)	Finnish_Swedish_100_	Finnish_Swedish_

Swedish (Sweden)	Finnish_Swedish_100_	Finnish_Swedish_
Syriac (Syria)	Syriac_100_[1]	Syriac_90_
Tajik (Tajikistan)	Cyrillic_General_100_	New to SQL Server 2008
Tamazight (Algeria, Latin)	Tamazight_100_	New to SQL Server 2008
Tamil (India)	Indic_General_100_[1]	Indic_General_90_
Tatar (Russia)	Tatar_100_	Tatar_90_, Cyrillic_General_
Telugu (India)	Indic_General_100_[1]	Indic_General_90_
Thai (Thailand)	Thai_100_	Thai_
Tibetan (PRC)	Tibetan_100_[1]	New to SQL Server 2008
Turkish (Turkey)	Turkish_100_	Turkish_
Turkmen (Turkmenistan)	Turkmen_100_	New to SQL Server 2008
Uighur (PRC)	Uighur_100_	New to SQL Server 2008
Ukrainian (Ukraine)	Ukrainian_100_	Ukrainian_
Upper Sorbian (Germany)	Upper_Sorbian_100_	New to SQL Server 2008
Urdu (Pakistan)	Urdu_100_	New to SQL Server 2008
Uzbek (Uzbekistan, Cyrillic)	Cyrillic_General_100_	Cyrillic_General_

Uzbek (Uzbekistan, Latin)	Uzbek_Latin_100_	Uzbek_Latin_90_
Vietnamese (Vietnam)	Vietnamese_100_	Vietnamese_
Welsh (United Kingdom)	Welsh_100_	New to SQL Server 2008
Wolof (Senegal)	French_100_	New to SQL Server 2008
Xhosa/isiXhosa (South Africa)	Latin1_General_100_	New to SQL Server 2008
Yakut (Russia)	Yakut_100_	New to SQL Server 2008
Yi (PRC)	Latin1_General_100_	New to SQL Server 2008
Yoruba (Nigeria)	Latin1_General_100_	New to SQL Server 2008
Zulu/isiZulu (South Africa)	Latin1_General_100_	New to SQL Server 2008
Deprecated, not available at server level in SQL Server 2008	Hindi	Hindi
Deprecated, not available at server level in SQL Server 2008	Korean_Wansung_Unicode	Korean_Wansung_Unicode
Deprecated, not available at server level in SQL Server 2008	Lithuanian_Classic	Lithuanian_Classic

Deprecated, not available at server level in SQL Server 2008	Macedonian	Macedonian

SQL Server 2008 Integration Services locales and code pages

Supported locales for flat files

Afrikaans	Azeri (Cyrillic, Azerbaijan)	Czech
Afrikaans (South Africa)	Azeri (Latin, Azerbaijan)	Czech (Czech Republic)
Albanian	Bashkir (Russia)	Danish
Albanian (Albania)	Basque	Danish (Denmark)
Alsatian (France)	Basque (Basque)	Dari (Afghanistan)
Amharic (Ethiopia)	Belarusian	Divehi
Arabic	Belarusian (Belarus)	Divehi (Maldives)
Arabic (Algeria)	Bengali (Bangladesh)	Dutch
Arabic (Bahrain)	Bengali (India)	Dutch (Belgium)
Arabic (Egypt)	Bosnian (Cyrillic, Bosnia and Herzegovina)	Dutch (Netherlands)
Arabic (Iraq)	Bosnian (Latin, Bosnia and Herzegovina)	English
Arabic (Jordan)	Breton (France)	English (Australia)
Arabic (Kuwait)	Bulgarian	English (Belize)
Arabic (Lebanon)	Bulgarian (Bulgaria)	English (Canada)
Arabic (Libya)	Catalan	English (Caribbean)
Arabic (Morocco)	Catalan (Catalan)	English (India)
Arabic (Oman)	Chinese (Hong Kong S.A.R.)	English (Ireland)
Arabic (Qatar)	Chinese (Macao S.A.R.)	English (Jamaica)
Arabic (Saudi Arabia)	Chinese (People's Republic of China)	English (Malaysia)
Arabic (Syria)	Chinese (Simplified)	English (New Zealand)
Arabic (Tunisia)	Chinese (Singapore)	English (Republic of the Philippines)
Arabic (U.A.E.)	Chinese (Taiwan)	English (Singapore)
Arabic (Yemen)	Chinese (Traditional)	English (South Africa)
Armenian	Corsican (France)	English (Trinidad and Tobago)
Armenian (Armenia)	Croatian	English (United Kingdom)
Assamese (India)	Croatian (Croatia)	English (United States)
Azeri	Croatian (Latin, Bosnia and Herzegovina)	English (Zimbabwe)

Estonian	Greenlandic (Greenland)
Estonian (Estonia)	Gujarati
Faroese	Gujarati (India)
Faroese (Faroe Islands)	Hausa (Latin, Nigeria)
Filipino (Philippines)	Hebrew
Finnish	Hebrew (Israel)
Finnish (Finland)	Hindi
French	Hindi (India)
French (Belgium)	Hungarian
French (Canada)	Hungarian (Hungary)
French (France)	Icelandic
French (Luxembourg)	Icelandic (Iceland)
French (Principality of Monaco)	Igbo (Nigeria)
French (Switzerland)	Indonesian
Frisian (Netherlands)	Indonesian (Indonesia)
Galician	Inuktitut (Latin, Canada)
Galician (Galician)	Inuktitut (Syllabics, Canada)
Georgian	Invariant Language (Invariant Country)
Georgian (Georgia)	Irish (Ireland)
German	isiXhosa (South Africa)
German (Austria)	isiZulu (South Africa)
German (Germany)	Italian
German (Liechtenstein)	Italian (Italy)
German (Luxembourg)	Italian (Switzerland)
German (Switzerland)	Japanese
Greek	Japanese (Japan)
Greek (Greece)	Kannada

Kannada (India)
Kazakh
Kazakh (Kazakhstan)
Khmer (Cambodia)
K'iche (Guatemala)
Kinyarwanda (Rwanda)
Kiswahili
Kiswahili (Kenya)
Konkani
Konkani (India)
Korean
Korean (Korea)
Kyrgyz
Kyrgyz (Kyrgyzstan)
Lao (Lao P.D.R.)
Latvian
Latvian (Latvia)
Lithuanian
Lithuanian (Lithuania)
Lower Sorbian (Germany)
Luxembourgish (Luxembourg)
Macedonian
Macedonian (Former Yugoslav Republic of Macedonia)
Malay
Malay (Brunei Darussalam)
Malay (Malaysia)
Malayalam (India)
Maltese (Malta)
Maori (New Zealand)
Mapudungun (Chile)
Marathi
Marathi (India)
Mohawk (Mohawk)
Mongolian
Mongolian (Cyrillic, Mongolia)
Mongolian (Traditional Mongolian, PRC)
Nepali (Nepal)
Norwegian
Norwegian, Bokmål (Norway)
Norwegian, Nynorsk (Norway)
Occitan (France)
Oriya (India)
Pashto (Afghanistan)
Persian
Persian (Iran)
Polish
Polish (Poland)
Portuguese
Portuguese (Brazil)
Portuguese (Portugal)
Punjabi
Punjabi (India)
Quechua (Bolivia)
Quechua (Ecuador)

Quechua (Peru)
Romanian
Romanian (Romania)
Romansh (Switzerland)
Russian
Russian (Russia)
Sami, Inari (Finland)
Sami, Lule (Norway)
Sami, Lule (Sweden)
Sami, Northern (Finland)
Sami, Northern (Norway)
Sami, Northern (Sweden)
Sami, Skolt (Finland)
Sami, Southern (Norway)
Sami, Southern (Sweden)
Sanskrit
Sanskrit (India)
Scottish Gaelic (United Kingdom)
Serbian
Serbian (Cyrillic, Bosnia and Herzegovina)
Serbian (Cyrillic, Montenegro)
Serbian (Cyrillic, Serbia and Montenegro (Former))
Serbian (Cyrillic, Serbia)
Serbian (Latin, Bosnia and Herzegovina)
Serbian (Latin, Montenegro)
Serbian (Latin, Serbia and Montenegro (Former))
Serbian (Latin, Serbia)
Sesotho sa Leboa (South Africa)
Setswana (South Africa)
Sinhala (Sri Lanka)
Slovak
Slovak (Slovakia)
Slovenian
Slovenian (Slovenia)
Spanish
Spanish (Argentina)
Spanish (Bolivia)
Spanish (Chile)
Spanish (Colombia)
Spanish (Costa Rica)
Spanish (Dominican Republic)
Spanish (Ecuador)
Spanish (El Salvador)
Spanish (Guatemala)
Spanish (Honduras)
Spanish (Mexico)
Spanish (Nicaragua)
Spanish (Panama)
Spanish (Paraguay)
Spanish (Peru)
Spanish (Puerto Rico)
Spanish (Spain)
Spanish (United States)
Spanish (Uruguay)

Spanish (Venezuela)
Swedish
Swedish (Finland)
Swedish (Sweden)
Syriac
Syriac (Syria)
Tajik (Cyrillic, Tajikistan)
Tamazight (Latin, Algeria)
Tamil
Tamil (India)
Tatar
Tatar (Russia)
Telugu
Telugu (India)
Thai
Thai (Thailand)
Tibetan (PRC)
Turkish
Turkish (Turkey)
Turkmen (Turkmenistan)
Ukrainian
Ukrainian (Ukraine)
Upper Sorbian (Germany)
Urdu
Urdu (Islamic Republic of Pakistan)
Uyghur (PRC)
Uzbek

Uzbek (Cyrillic, Uzbekistan)
Uzbek (Latin, Uzbekistan)
Vietnamese
Vietnamese (Vietnam)
Welsh (United Kingdom)
Wolof (Senegal)
Yakut (Russia)
Yi (PRC)
Yoruba (Nigeria)

Supported code pages for flat files

10000 (MAC - Roman)
10001 (MAC - Japanese)
10002 (MAC - Traditional Chinese Big5)
10003 (MAC - Korean)
10004 (MAC - Arabic)
10005 (MAC - Hebrew)
10006 (MAC - Greek I)
10007 (MAC - Cyrillic)
10008 (MAC - Simplified Chinese GB 2312)
10010 (MAC - Romania)
10017 (MAC - Ukraine)
10021 (MAC - Thai)
10029 (MAC - Latin II)
10079 (MAC - Icelandic)
10081 (MAC - Turkish)
10082 (MAC - Croatia)
1026 (IBM EBCDIC - Turkish (Latin-5))
1047 (IBM EBCDIC - Latin-1/Open System)
1140 (IBM EBCDIC - U.S./Canada (37 + Euro))
1141 (IBM EBCDIC - Germany (20273 + Euro))
1142 (IBM EBCDIC - Denmark/Norway (20277 + Euro))
1143 (IBM EBCDIC - Finland/Sweden (20278 + Euro))
1144 (IBM EBCDIC - Italy (20280 + Euro))
1145 (IBM EBCDIC - Latin America/Spain (20284 + Euro))
1146 (IBM EBCDIC - United Kingdom (20285 + Euro))
1148 (IBM EBCDIC - International (500 + Euro))
1149 (IBM EBCDIC - Icelandic (20871 + Euro))
1250 (ANSI - Central Europe)
1251 (ANSI - Cyrillic)
1252 (ANSI - Latin I)
1253 (ANSI - Greek)
1254 (ANSI - Turkish)
1255 (ANSI - Hebrew)
1256 (ANSI - Arabic)
1257 (ANSI - Baltic)
1258 (ANSI/OEM - Viet Nam)
1361 (Korean - Johab)
20000 (CNS - Taiwan)
20001 (TCA - Taiwan)
20002 (Eten - Taiwan)
20003 (IBM5550 - Taiwan)
20004 (TeleText - Taiwan)
20005 (Wang - Taiwan)
20105 (IA5 IRV International Alphabet No.5)
20106 (IA5 German)
20107 (IA5 Swedish)
20108 (IA5 Norwegian)
20127 (US-ASCII)
20261 (T.61)
20269 (ISO 6937 Non-Spacing Accent)
20273 (IBM EBCDIC - Germany)
20277 (IBM EBCDIC - Denmark/Norway)
20278 (IBM EBCDIC - Finland/Sweden)
20280 (IBM EBCDIC - Italy)
20284 (IBM EBCDIC - Latin America/Spain)
20285 (IBM EBCDIC - United Kingdom)
20290 (IBM EBCDIC - Japanese Katakana Extended)
20297 (IBM EBCDIC - France)
20420 (IBM EBCDIC - Arabic)
20423 (IBM EBCDIC - Greek)

20424 (IBM EBCDIC - Hebrew)
20833 (IBM EBCDIC - Korean Extended)
20838 (IBM EBCDIC - Thai)
20866 (Russian - KOI8)
20871 (IBM EBCDIC - Icelandic)
20880 (IBM EBCDIC - Cyrillic (Russian))
20905 (IBM EBCDIC - Turkish)
20924 (IBM EBCDIC - Latin-1/Open System (1047 + Euro))
20932 (JIS X 0208-1990 & 0212-1990)
20936 (Simplified Chinese GB2312)
21025 (IBM EBCDIC - Cyrillic (Serbian, Bulgarian))
21027 (Ext Alpha Lowercase)
21866 (Ukrainian - KOI8-U)
28591 (ISO 8859-1 Latin I)
28592 (ISO 8859-2 Central Europe)
28593 (ISO 8859-3 Latin 3)
28594 (ISO 8859-4 Baltic)
28595 (ISO 8859-5 Cyrillic)
28596 (ISO 8859-6 Arabic)
28597 (ISO 8859-7 Greek)
28598 (ISO 8859-8 Hebrew: Visual Ordering)
28599 (ISO 8859-9 Latin 5)
28603 (ISO 8859-13 Latin 7)
28605 (ISO 8859-15 Latin 9)
37 (IBM EBCDIC - U.S./Canada)
38598 (ISO 8859-8 Hebrew: Logical Ordering)
437 (OEM - United States)
500 (IBM EBCDIC - International)
50220 (ISO-2022 Japanese with no halfwidth Katakana)
50221 (ISO-2022 Japanese with halfwidth Katakana)
50222 (ISO-2022 Japanese JIS X 0201-1989)
50225 (ISO-2022 Korean)
50227 (ISO-2022 Simplified Chinese)
50229 (ISO-2022 Traditional Chinese)
51949 (EUC-Korean)
52936 (HZ-GB2312 Simplified Chinese)
54936 (GB18030 Simplified Chinese)
57002 (ISCII - Devanagari)
57003 (ISCII - Bengali)
57004 (ISCII - Tamil)
57005 (ISCII - Telugu)
57006 (ISCII - Assamese)
57007 (ISCII - Oriya)
57008 (ISCII - Kannada)
57009 (ISCII - Malayalam)
57010 (ISCII - Gujarati)
57011 (ISCII - Punjabi (Gurmukhi))
65000 (UTF-7)
65001 (UTF-8)
708 (Arabic - ASMO)
720 (Arabic - Transparent ASMO)
737 (OEM - Greek 437G)
775 (OEM - Baltic)
850 (OEM - Multilingual Latin I)
852 (OEM - Latin II)
855 (OEM - Cyrillic)
857 (OEM - Turkish)
858 (OEM - Multilingual Latin I + Euro)
860 (OEM - Portuguese)
861 (OEM - Icelandic)

862	(OEM - Hebrew)
863	(OEM - Canadian French)
864	(OEM - Arabic)
865	(OEM - Nordic)
866	(OEM - Russian)
869	(OEM - Modern Greek)
870	(IBM EBCDIC - Multilingual/ROECE (Latin-2))
874	(ANSI/OEM - Thai)
875	(IBM EBCDIC - Modern Greek)
932	(ANSI/OEM - Japanese Shift-JIS)
936	(ANSI/OEM - Simplified Chinese GBK)
949	(ANSI/OEM - Korean)
950	(ANSI/OEM - Traditional Chinese Big5)

Windows/SAP code pages mappings

2-char Lang ID	1-char Lang ID	Language	Win32 code page	SAP code page	ISO character set
ar	a	Arabic	1256	1100	Windows-1256

he	b	Hebrew	1255	1800	Windows-1255
cs	c	Czech	1250	1404	Windows-1250
de	d	German	1252	1100	us-ascii
en	e	English	1252	1100	us-ascii
fr	f	French	1252	1100	us-ascii
el	g	Greek	1253	1700	Windows-1253
hu	h	Hungarian	1250	1401	Windows-1250
it	i	Italian	1252	1100	us-ascii
ja	j	Japanese	932	8000	x-sjis
da	k	Danish	1252	1100	Windows-1250
pl	l	Polish	1250	1401	Windows-1250
zf	m	Traditional Chinese	950	8300	big5
nl	n	Dutch	1252	1100	us-ascii
no	o	Norwegian	1252	1100	us-ascii
pt	p	Portuguese	1252	1100	us-ascii
sk	q	Slovakian	1250	1404	Windows-1250
ru	r	Russian	1251	1500	Windows-1251
es	s	Spanish	1252	1100	us-ascii
tr	t	Turkish	1254	1610	Windows-1254
fi	u	Finnish	1252	1100	us-ascii
sv	v	Swedish	1252	1100	us-ascii
bg	w	Bulgarian	1251	1500	Windows-1251
zh	1	Simplified Chinese	936	8400	GB_2312_80
th	2	Thai	874	8600	x-TIS-620
ko	3	Korean	949	8500	ks_c_5601-1987
ro	4	Romanian	1250	1401	Windows-1250
sl	5	Slovenian	1250	1401	Windows-1250
hr	6	Croatian	1250	1401	Windows-1250
zz	z	Language-independent	1252	1100	Z