
# SQL Anywhere® Studio and Unicode

*A whitepaper from iAnywhere Solutions, Inc., a subsidiary of Sybase, Inc.*

# Contents

# Does SQL Anywhere Studio support Unicode?

SQL Anywhere Studio supports Unicode. In particular:

- Adaptive Server Anywhere supports the UTF-8 character encoding for Unicode, using the UTF8 collation.
- The Adaptive Server Anywhere ODBC driver, OLE DB provider, and ADO.NET Data Provider are Unicode-enabled.
- Adaptive Server Anywhere provides character set conversion between the database and client character sets, supporting both Unicode and non-Unicode applications.
- Adaptive Server Anywhere supports the SORTKEY and COMPARE functions, as well as the SORT_COLLATION database option, providing advanced ordering and comparison of Unicode and non-Unicode strings.
- MobiLink is a Unicode application.

# What is Unicode?

Unicode is a standard for encoding written characters and text, allowing for storage and international exchange of text data. Unicode allows for representation of over one million characters and symbols in a universal character encoding scheme. The Unicode Standard and its associated documents describe the mapping between characters and numeric values, but also describe:

- encoding schemes
- compression
- sorting and comparison
- many other issues related to storage and exchange of international data

At the time of writing, the Unicode Standard 4.0 had just been released.

Unicode has three encoding forms: UTF-8, UTF-16, and UTF-32. Other encoding forms, such as UCS-2 and UCS-4, no longer conform to the Unicode Standard.

**UTF-16**

- Unicode Transformation Format 16-bit.
- Most common characters use one 16-bit value. This set of characters was once called UCS-2 (Universal Character Set 2 Octet Form), and came to be viewed as being Unicode. However, conformant 16-bit Unicode implementations now use UTF-16.

- Rare supplementary characters are represented using a pair of 16-bit values, called a surrogate pair.
- Who uses UTF-16 today?
  - Windows NT/2000/XP
  - Windows CE
  - Java
  - Mac OS X

**UTF-32**

- Unicode Transformation Format 32-bit.
- 32-bit fixed-width characters.
- UTF-32 is commonly used as a processing form in UNIX implementations.
- Equivalent to UCS-4 (Universal Character Set 4 Octet Form).

The UTF-16 and UTF-32 encodings are difficult for traditional C/C++ programs to handle since standard “char” type processing will not work.

- A char is eight bits. A wide char type must be used instead, often resulting in significant changes to a typical program.
- A traditional program, using char for its character handling, cannot process UTF-16 or UTF-32 characters correctly, for example, if they are encountered in a file.

Because of the difficulty of handling UTF-16 and UTF-32 characters in traditional C/C++ programs, the Unicode Standard also provides an 8-bit, variable-width encoding that is compatible with use of the char data type. This encoding is relatively easy for C/C++ programs and for string processing code and protocols to handle.

**UTF-8**

- Unicode Transformation Format 8-bit.
- Multi-byte encoding for Unicode. Each character consists of one to four bytes.
- ASCII characters (0x00–0x7F) are represented in UTF-8 by a single byte with the same value as ASCII.
  - Storing plain English text in UTF-8 does not require any additional space.
- ASCII characters (0x00–0x7F) do not appear within a two-, three-, or four-byte character.
  - Simple char processing of strings, such as searching for delimiters or determining the string length by searching for the null character, works without modification.
- Who uses UTF-8 today?
  - Databases
  - Internet protocols

- World Wide Web (commonly used for HTML web pages)
- UNIX

For programs that do not perform significant manipulation of the content of strings, it may be possible to use UTF-8 strings with little or no modification. More extensive modifications may be required if the program splits strings, does substring matching, or is otherwise concerned with any character value outside the ASCII range. In this case, the program must be modified to respect character boundaries (as opposed to byte boundaries).

# Adaptive Server Anywhere database server support for Unicode

Adaptive Server Anywhere support for Unicode includes the following features:

- the UTF8 collation
- the SORTKEY function
- the COMPARE function
- the SORT_COLLATION database option
- character set conversion

## The UTF8 collation

The UTF8 collation allows the storage of any UTF-8 character, up to four bytes in length. ASCII characters are encoded as one byte, so there is no overhead when storing only English characters. Accented and other European characters are encoded as two bytes. Most Asian language characters are encoded as three bytes. Four-byte characters are used for Unicode supplementary characters, which are rare.

Sorting the UTF8 collation encounters a limitation in Adaptive Server Anywhere. Adaptive Server Anywhere sorting using built-in collations, including UTF8, is based only on the first byte of a multi-byte character. For the UTF8 collation, the ordering is not meaningful to people. For example, European accented characters are typically stored as two bytes. Many accented characters may share the same first byte. Since Adaptive Server Anywhere sorting looks at only the first byte, it is not possible for Adaptive Server Anywhere to properly group accented and unaccented characters with the same base letter. The typical result would group a variety of accented characters together in an order unrelated to, and probably distant from, their base letter. For this reason, Adaptive Server Anywhere supports the SORTKEY function and the SORT_COLLATION database option.

## The SORTKEY function

The Adaptive Server Anywhere database server provides the SORTKEY function, a function that is also provided in Sybase Adaptive Server Enterprise. SORTKEY builds a multipart binary sort key for a string value, consisting of the base letters, followed by accent values, followed by case values. The corresponding ordering is based on the Unicode Collation Algorithm, and is compatible with ISO 14651.

Comparisons based on the “default” ordering work as described in the list below. Adaptive Server Anywhere also provides orderings that ignore accent and case differences. This description uses the word “equivalent”, rather than “equal”, to allow for many variations in what is considered to be equal. The details of how exact a match must be to be considered “equivalent” or “equal” is beyond the scope of this document.

1. For letters, the letter is separated into three components: the base letter, the accent value of the letter, and the case (uppercase or lowercase). Ligatures (for example, ß, the German sharp s letter) are converted into two letters (in this case, ss). The base letter is used as the base character in the following steps.
2. For non-letter characters, the character itself is used as the base character.
3. The values are compared initially using only the base characters. If the base characters are not equivalent, then the strings are not equivalent and the comparison is complete.
4. If the base letters are equivalent, the accent values are examined. If the accent values are not equivalent, then the strings are not equivalent and the comparison is complete. The ordering of accented letters is beyond the scope of this document.
5. If the base letters and all other characters are equivalent, and the accents are equivalent, letter case is examined. If the letter case is different, then the original values are different. If the letter case is the same, then the original values are equivalent. Lowercase letters are considered “less than” uppercase letters.

SORTKEY is provided as a built-in function in SQL Anywhere Studio 8.0.0 and higher. It is also available for SQL Anywhere Studio 7 as a collection of stored procedures, external functions, and a DLL.

### Using sort keys

There are two standard ways of using sort keys. They can be created dynamically, or they can be stored in the database.

Creating sort keys dynamically can be useful when small database size is important, and when the number and size of ORDER BY operations is small. The following query dynamically creates a sort key:

```
SELECT Name FROM T ORDER BY SORTKEY(Name)
```

The results of this query are sorted using a dynamically created sort key for the Name column. The sort keys will be generated, the values will be sorted, and

then the sort keys will be discarded. Database size is not affected by this method of using sort keys, however, performance may be adversely affected because of the need to create and destroy the sort keys each time the statement is executed.

☞ For information about automatically using SORTKEY without changing the database schema or the SQL statements used, see "SORT_COLLATION database option" on page 7.

For performance reasons, storing sort keys in the database is recommended in most cases. A simple method is to create shadow columns and then use the COMPUTE clause to populate the shadow columns with the sort key value for the column to be sorted. For example,

```
CREATE TABLE T (
      Name      CHAR(100),
      Sh_Name   LONG BINARY COMPUTE(SORTKEY(Name))
);
INSERT INTO T (Name) VALUES ('François Allaire');
```

The INSERT statement will automatically compute the sort key for the Name column.

The following SELECT statements both achieve the same result.

```
SELECT Name
FROM T ORDER BY Sh_Name;
```

```
SELECT Name
FROM T ORDER BY SORTKEY(Name);
```

In each case, the ORDER BY clause uses the Sh_Name shadow column for the sort. The SORT_COLLATION database option can be used to cause SORTKEY to be applied to all ORDER BY operations on character columns.

The default ordering for SORTKEY is a multilingual ordering, which is sufficient for most users.

☞ For a complete list of SORTKEY orderings, refer to your SQL Anywhere Studio documentation.

## The COMPARE function

The COMPARE function is closely related to the SORTKEY function. The COMPARE function takes two character parameters, generates sort keys for the parameters, and compares the sort keys. The values are compared using the same rules as an ORDER BY clause based on the SORTKEY function. If the values are equal, the function returns zero. "Less than" returns –1, and "greater than" returns 1. Consider the following query:

```
SELECT Name
FROM T
WHERE COMPARE( Name, 'Straße', 'noaccent' ) = 0;
```

If Name is Strasse, the COMPARE function will return 0 (equal).

In contrast,

```
SELECT Name
FROM T WHERE Name = 'Straße';
```

will return 1 (greater than) because the equality operator uses the default comparison functionality, which treats the ligature ß as a single s, rather than ss.

## The SORT_COLLATION database option

The SORT_COLLATION database option causes SORTKEY to be applied to all ORDER BY operations on character columns, without requiring changes to the database schema or SQL statements. For performance reasons, a schema change may be recommended so that the sort keys for character columns are generated and stored with the table, rather than generated and discarded each time that the ORDER BY operation occurs.

SORT_COLLATION can be applied to all users, to a particular user, or to the current connection. For example,

```
SET TEMPORARY OPTION SORT_COLLATION='default';
```

causes all ORDER BY operations on character types to apply the default SORTKEY ordering for the duration of the current connection. For example,

```
SELECT Name
FROM T
ORDER BY Name;
```

becomes equivalent to

```
SELECT Name
FROM T
ORDER BY SORTKEY(Name);
```

and to

```
SELECT Name
FROM T
ORDER BY SORTKEY(Name, 'default');
```

## Database character types

An Adaptive Server Anywhere database uses CHAR, VARCHAR, and LONG VARCHAR types to store all character data. The type length is the number of bytes, not the number of characters.

### Character set conversion

An application can use one of a wide variety of character sets, however, the character set used is usually the native character set for the machine. The term code page is often used interchangeably with character set. An English Windows application would typically use the character set cp1252. Japanese Windows

applications would typically use the character set cp932.

Unicode applications are becoming more common. An Adaptive Server Anywhere-based application may use Unicode, depending on the Adaptive Server Anywhere client interface.

☞ For information, see "Client Library Support for Unicode" on page 8.

The database server may be running a database that uses a different character set from that of the client. For example, the database character set may be UTF-8. In this case, it is necessary for character set conversion to take place between the client and server. This conversion can take place within the client application, within the client library, or within the database server. The client and the server negotiate where character set conversion should occur.

Character set conversion is performed only if necessary. For example, if the application character set and the database character set are the same, the database server will not perform character set conversion for connections from that application.

The application may specify the application character set for single- and multi-byte characters using the "charset=" connection parameter. For Windows applications, if the character set is not specified, it is assumed to be the Windows character set, also known as the ANSI Code Page, or ACP.

Character set conversion on the server is controlled by the –ct database server option. In SQL Anywhere Studio version 8 and higher, character set conversion is ON by default. In earlier versions, character set conversion is OFF by default.

If character set conversion is turned on, character set conversion is affected by the database character set and by the application character set. If character set conversion is turned off, the server relies on the application or client library to communicate using the database character set.

## Client library support for Unicode

### ODBC driver support for Unicode

On Windows, the ODBC client library is Unicode-enabled, and supports wide characters (UTF-16) using the SQL_WCHAR type. The Microsoft ODBC Driver Manager uses the ODBC driver's Unicode API, causing character set conversion to occur on the client side.

An application can be compiled with the UNICODE macro defined. In this case, all strings specified to ODBC functions are Unicode strings. The Unicode API in the ODBC Driver Manager will be used, and the Driver Manager will pass the Unicode strings to and from the ODBC driver without modification. The ODBC driver will perform character set conversion between Unicode and the database character set.

If an application is not compiled with the UNICODE macro defined, the ANSI API in the ODBC Driver Manager is used. The Driver Manager will convert all strings from the application character set to Unicode when calling the ODBC driver's

Unicode API. Returned strings will be converted from Unicode to the application's character set. The ODBC Driver Manager assumes that the application character set is the Windows character set.

Bound variables are not affected by Driver Manager Unicode conversion, but may be converted by the ODBC driver or by the database server. Bound variables of type SQL_CHAR are assumed to be in the Windows character set (matching the behavior of the ODBC Driver Manager).

The ODBC driver negotiates with the database server to determine how character set conversion should be handled. Since the Unicode API to the ODBC driver is being used, the ODBC driver will convert all Unicode strings to and from the character set required by the database. The ODBC driver negotiates with the database server to minimize the number of character set conversions that are required. Depending on the outcome of the negotiation, character set conversion may take place in either the ODBC driver or the database server.

### OLE DB provider support for Unicode

The OLE DB provider supports Unicode (UTF-16) characters. As with the ODBC driver, the OLE DB provider and the database server negotiate character set conversion.

### ADO.NET data provider support for Unicode

The ADO.NET data provider supports only Unicode (UTF-16) characters. The ADO.NET data provider handles conversion between Unicode and the database character set.

### Embedded SQL support for Unicode

The embedded SQL preprocessor and client library do not support the UTF-16 encoding. If the application uses UTF-16 characters, it is the application's responsibility to convert the characters to and from the required database character set, usually UTF-8. For example, the application must inform the server that its character set is UTF-8, using the connection parameter `charset=utf8`.

Conversion between UTF-16 and UTF-8 is efficient. No tables are required as the transformation is a straightforward rearrangement of bits. On Windows systems, the WideCharToMultiByte and MultiByteToWideChar functions may be used for this conversion.

### Java support for Unicode

Java uses the Unicode (UTF-16) character encoding. To communicate with an Adaptive Server Anywhere database, Java applications use JDBC. There are two methods for communicating with Adaptive Server Anywhere using JDBC:

- **iAnywhere JDBC driver** The iAnywhere JDBC driver is a layer of Java code that provides an interface between JDBC and the native ODBC driver. When

using the iAnywhere JDBC driver, character set conversion is handled as described in the section on ODBC.

☞ For more information, see "ODBC driver support for Unicode" on page 8.

- **jConnect** jConnect is a pure Java JDBC implementation from Sybase. jConnect is provided with SQL Anywhere Studio and uses the TDS communication protocol. jConnect determines the database character set and performs the required character set conversion.

## MobiLink support for Unicode

The MobiLink synchronization server is a Unicode application.

For Windows CE databases, the upload and download streams use Unicode (UTF-16) character encoding. For other remote platforms, uploads and downloads take place in the character set of the remote database.

MobiLink communicates with the consolidated database using ODBC's Unicode API. If necessary, MobiLink performs conversion between Unicode and the character set required by the remote database.

## UltraLite support for Unicode

Java uses the Unicode (UTF-16) character encoding. No character set conversion is required.

C/C++ applications can be written using the UTF-16 character encoding. In this case, the UNICODE macro should be defined when compiling on Windows. If UNICODE is defined, the UL_CHAR type changes to 16-bit characters. The UltraLite data store supports the UTF-8 character encoding.

# Legal Notice



## Contact Us

**iAnywhere Solutions Worldwide Headquarters** One Sybase Drive, Dublin, CA, 94568 USA

**Phone** 1-800-801-2069 (in US and Canada)

**Fax** 1-519-747-4971

**World Wide Web** *http://www.ianywhere.com*

**E-mail** contact.us@ianywhere.com