Understanding Unicode and ODBC Data Access

DataDirect Connect® Series for ODBC Drivers

Introduction

This document provides a brief background on Unicode, its development, and how it is accommodated by Unicode and non-Unicode DataDirect Connect® Series *for* ODBC drivers. The DataDirect Connect Series *for* ODBC drivers include DataDirect Connect and Connect XE *for* ODBC as well as DataDirect Connect64® and Connect64 XE *for* ODBC.

Character Encoding

Most developers know that Unicode is a standard encoding that can be used to support multi-lingual character sets. Unfortunately, understanding Unicode is not as simple as its name would indicate. Software developers have used a number of character encodings, from ASCII to Unicode, to solve the many problems that arise when developing software applications that can be used worldwide.

Background

Most legacy computing environments have used ASCII character encoding developed by the ANSI standards body to store and manipulate character strings inside software applications. ASCII encoding was convenient for programmers because each ASCII character could be stored as a byte. The initial version of ASCII used only 7 of the 8 bits available in a byte, which meant that applications could use only 128 different characters. This version of ASCII could not account for European characters, and was completely inadequate for Asian characters. Using the eighth bit to extend the total range of characters to 256 added support for most European characters. Today, ASCII refers to either the 7-bit or 8-bit encoding of characters.

As the need increased for applications with additional international support, ANSI again increased the functionality of ASCII by developing an extension to accommodate multilingual software. The extension, known as the Double-Byte Character Set (DBCS), allowed existing applications to function without change, but provided for the use of additional characters, including complex Asian characters. With DBCS, characters map to either one byte (for example, American ASCII characters) or two bytes (for example, Asian characters). The DBCS environment also introduced the concept of an operating system code page that identified how characters would be

encoded into byte sequences in a particular computing environment. DBCS encoding provided a cross-platform mechanism for building multilingual applications.

The DataDirect Connect Series *for* ODBC UNIX and Linux drivers can use double-byte character sets. The drivers normally use the character set defined by the default locale "C" unless explicitly pointed to another character set. The default locale "C" corresponds to the 7-bit US-ASCII character set. Use the following procedure to set the locale to a different character set:

1. Add the following line at the very beginning of applications that use double-byte character sets:

   ```
   setlocale (LC_ALL, "");
   ```

   This is a standard UNIX function. It selects the character set indicated by the environment variable LANG as the one to be used by X/Open compliant character handling functions. If this line is not present, or if LANG is either not set or is set to NULL, the default locale "C" is used.

2. Set the LANG environment variable to the appropriate character set. The UNIX command `locale -a` can be used to display all supported character sets on your system.

For more information, refer to the man pages for "locale" and "setlocale."

Using a DBCS, however, was not ideal; many developers felt that there was a better way to solve the problem. A group of leading software companies joined forces to form the Unicode Consortium. Together, they produced a new solution to building worldwide applications—Unicode. Unicode was originally designed as a fixed-width, uniform two-byte designation that could represent all modern scripts without the use of code pages. The Unicode Consortium has continued to evaluate new characters, and the current number of supported characters is over 95,000.

Although it seemed to be the perfect solution to building multilingual applications, Unicode started off with a significant drawback—it would have to be retrofitted into existing computing environments. To use the new paradigm, all applications would have to change. As a result, several standards-based transliterations were designed to convert two-byte fixed Unicode values into more appropriate character encodings, including, among others, UTF-8, UCS-2, and UTF-16.

UTF-8 is a standard method for transforming Unicode values into byte sequences that maintain transparency for all ASCII codes. UTF-8 is recognized by the Unicode Consortium as a mechanism for transforming Unicode values and is popular for use with HTML, XML, and other protocols. UTF-8 is, however, currently used primarily on AIX, HP-UX, Solaris, and Linux.

UCS-2 encoding is a fixed, two-byte encoding sequence and is a method for transforming Unicode values into byte sequences. It is the standard for Windows 95, Windows 98, Windows Me, and Windows NT.

UTF-16 is a superset of UCS-2, with the addition of some special characters in surrogate pairs. UTF-16 is the standard encoding for Windows 2000, Windows XP, Windows Server 2003, and Windows Vista.

Unicode Support in Databases

Recently, database vendors have begun to support Unicode data types natively in their systems. With Unicode support, one database can hold multiple languages. For example, a large multinational corporation could store expense data in the local languages for the Japanese, U.S., English, German, and French offices in one database.

Not surprisingly, the implementation of Unicode data types varies from vendor to vendor. For example, the Microsoft SQL Server 2000 implementation of Unicode provides data in UTF-16 format, while Oracle provides Unicode data types in UTF-8 and UTF-16 formats. A consistent implementation of Unicode not only depends on the operating system, but also on the database itself.

Unicode Support in ODBC

Prior to the ODBC 3.5 standard, all ODBC access to function calls and string data types was through ANSI encoding (either ASCII or DBCS). Applications and drivers were both ANSI-based.

The ODBC 3.5 standard specified that the ODBC Driver Manager (on both Windows and UNIX) be capable of mapping both Unicode function calls and string data types to ANSI encoding as transparently as possible. This meant that ODBC 3.5-compliant Unicode applications could use Unicode function calls and string data types with ANSI drivers because the Driver Manager could convert them to ANSI. Because of character limitations in ANSI, however, not all conversions are possible.

The ODBC Driver Manager version 3.5 and later, therefore, supports the following configurations:

- ANSI application with an ANSI driver
- ANSI application with a Unicode driver
- Unicode application with a Unicode driver
- Unicode application with an ANSI driver

A Unicode application can work with an ANSI driver because the Driver Manager provides limited Unicode-to-ANSI mapping. The Driver Manager makes it possible for a pre-3.5 ANSI driver to work with a Unicode application. What distinguishes a Unicode driver from a non-Unicode driver is the Unicode driver's capacity to interpret Unicode function calls without the intervention of the Driver Manager, as described in the following section.

Unicode and Non-Unicode ODBC Drivers

The way in which a driver handles function calls from a Unicode application determines whether it is called a "Unicode driver."

Function Calls

Instead of the standard ANSI SQL function calls, such as SQLConnect, Unicode applications use "W" (wide) function calls, such as SQLConnectW. If the driver is a true Unicode driver, it can understand "W" function calls and the Driver Manager can pass them through to the driver without conversion to ANSI.

If the driver is a non-Unicode driver, it cannot understand W function calls, and the Driver Manager must convert them to ANSI calls before sending them to the driver. The Driver Manager determines the ANSI encoding system to which it must convert by referring to a code page. On Windows, this reference is to the Active Code Page. On UNIX and Linux, it is to the IANAAppCodePage connection string attribute, part of the odbc.ini file.

The following examples illustrate these conversion streams for the DataDirect Connect Series *for* ODBC drivers. The Driver Manager on UNIX and Linux prior to the DataDirect Connect Series *for* ODBC Release 5.0 assumes that Unicode applications and Unicode drivers use the same encoding (UTF-8). For the DataDirect Connect Series *for* ODBC Release 5.0 and higher on UNIX and Linux, the Driver Manager determines the type of Unicode encoding of both the application and the driver, and performs conversions when the application and driver use different types of encoding. This determination is made by checking two ODBC environment attributes: SQL_ATTR_APP_UNICODE_TYPE and SQL_ATTR_DRIVER_UNICODE_TYPE. The Driver Manager and Unicode Encoding on UNIX describes in detail how this is done.

Unicode Application with a Non-Unicode Driver

An operation involving a Unicode application and a non-Unicode driver incurs more overhead because function conversion is involved.

Windows

1. The Unicode application sends UCS-2/UTF-16 function calls to the Driver Manager.
2. The Driver Manager converts the function calls from UCS-2/UTF-16 to ANSI. The type of ANSI is determined by the Driver Manager through reference to the client machine's Active Code Page.
3. The Driver Manager sends the ANSI function calls to the non-Unicode driver.
4. The driver returns ANSI argument values to the Driver Manager.

5. The Driver Manager converts the function calls from ANSI to UCS-2/UTF-16 and returns these converted calls to the application.

UNIX and Linux: DataDirect Connect Series *for* ODBC Releases Prior to 5.0

1. The Unicode application sends UTF-8 function calls to the Driver Manager.
2. The Driver Manager converts the function calls from UTF-8 to ANSI. The type of ANSI is determined by the Driver Manager through reference to the client machine's value for the IANAAppCodePage connection string attribute.
3. The Driver Manager sends the converted ANSI function calls to the non-Unicode driver.
4. The driver returns ANSI argument values to the Driver Manager.
5. The Driver Manager converts the function calls from ANSI to UTF-8 and returns these converted calls to the application.

UNIX and Linux: DataDirect Connect Series *for* ODBC 5.0 and Higher

1. The Unicode application sends function calls to the Driver Manager. The Driver Manager expects these function calls to be UTF-8 or UTF-16 based on the value of the SQL_ATTR_APP_UNICODE_TYPE attribute.
2. The Driver Manager converts the function calls from either UTF-8 or UTF-16 to ANSI. The type of ANSI is determined by the Driver Manager through reference to the client machine's value for the IANAAppCodePage connection string attribute.
3. The Driver Manager sends the converted ANSI function calls to the non-Unicode driver.
4. The driver returns ANSI argument values to the Driver Manager.
5. The Driver Manager converts the function calls from ANSI to either UTF-8 or UTF-16 and returns these converted calls to the application.

Unicode Application with a Unicode Driver

An operation involving a Unicode application and a Unicode driver that use the same Unicode encoding is more efficient because no function conversion is involved. If the application and the driver each use different types of encoding, there is some conversion overhead. See The Driver Manager and Unicode Encoding on UNIX for details.

Windows

1. The Unicode application sends UCS-2 or UTF-16 function calls to the Driver Manager.
2. The Driver Manager does not have to convert the UCS-2/UTF-16 function calls to ANSI. It passes the Unicode function call to the Unicode driver.
3. The driver returns UCS-2/UTF-16 argument values to the Driver Manager.
4. The Driver Manager returns UCS-2/UTF-16 function calls to the application.

UNIX and Linux: DataDirect Connect Series *for* ODBC Releases Prior to 5.0

1. The Unicode application sends UTF-8 function calls to the Driver Manager.
2. The Driver Manager does not have to convert the UTF-8 function calls to ANSI. It passes the Unicode function call with UTF-8 arguments to the Unicode driver.
3. The driver returns UTF-8 argument values to the Driver Manager.
4. The Driver Manager returns UTF-8 function calls to the application.

UNIX and Linux: DataDirect Connect Series *for* ODBC 5.0 and Higher

1. The Unicode application sends function calls to the Driver Manager. The Driver Manager expects these function calls to be UTF-8 or UTF-16 based on the value of the SQL_ATTR_APP_UNICODE_TYPE attribute.
2. The Driver Manager passes Unicode function calls to the Unicode driver. The Driver Manager has to perform function call conversions if the SQL_ATTR_APP_UNICODE_TYPE is different from the SQL_ATTR_DRIVER_UNICODE_TYPE.
3. The driver returns argument values to the Driver Manager. Whether these are UTF-8 or UTF-16 argument values is based on the value of the SQL_ATTR_DRIVER_UNICODE_TYPE attribute.
4. The Driver Manager returns appropriate function calls to the application, based on the SQL_ATTR_APP_UNICODE_TYPE attribute. The Driver Manager has to perform function call conversions if the SQL_ATTR_DRIVER_UNICODE_TYPE value is different from the SQL_ATTR_APP_UNICODE_TYPE value.

Data

ODBC C data types are used to indicate the type of C buffers that store data in the application. This is in contrast to SQL data types, which are mapped to native database types to store data in a database (data store). ANSI applications bind to the C data type SQL_C_CHAR and expect to receive information bound in the same way. Similarly, most Unicode applications bind to the C data type SQL_C_WCHAR (wide data type) and expect to receive information bound in the same way. Any ODBC 3.5-compliant Unicode driver must be capable of supporting SQL_C_CHAR and SQL_C_WCHAR so that it can return data to both ANSI and Unicode applications.

When the driver communicates with the database, it must use ODBC SQL data types, such as SQL_CHAR and SQL_WCHAR, that map to native database types. In the case of ANSI data and an ANSI database, the driver receives data bound to SQL_C_CHAR and passes it to the database as SQL_CHAR. The same is true of SQL_C_WCHAR and SQL_WCHAR in the case of Unicode data and a Unicode database.

When data from the application and the data stored in the database differ in format, for example, ANSI application data and Unicode database data, conversions must be performed. The driver cannot receive SQL_C_CHAR data and pass it to a Unicode database that expects to receive a SQL_WCHAR data type. The driver or the Driver Manager must be capable of converting SQL_C_CHAR to SQL_WCHAR, and vice versa.

The simplest cases of data communication are when the application, the driver, and the database are all of the same type and encoding, ANSI-to-ANSI-to-ANSI or Unicode-to-Unicode-to-Unicode. There is no data conversion involved in these instances.

When a difference exists between data types, a conversion from one type to another must take place at the driver or Driver Manager level, which involves additional overhead. The type of driver determines whether these conversions are performed by the driver or the Driver Manager. The Driver Manager and Unicode Encoding on UNIX describes how the Driver Manager determines the type of Unicode encoding of the application and driver.

The following sections discuss two basic types of data conversion in the DataDirect Connect Series *for* ODBC drivers and the Driver Manager. How an individual driver exchanges different types of data with a particular database at the database level is beyond the scope of this discussion.

Unicode Driver

The Unicode driver, not the Driver Manager, must convert SQL_C_CHAR (ANSI) data to SQL_WCHAR (Unicode) data, and vice versa, as well as SQL_C_WCHAR (Unicode) data to SQL_CHAR (ANSI) data, and vice versa.

The driver must use client code page information (Active Code Page on Windows and IANAAppCodePage connection string attribute on UNIX/Linux) to determine which ANSI code page to use for the conversions. The Active Code Page or IANAAppCodePage must match the database default character encoding; if it does not, conversion errors are possible.

ANSI Driver

The Driver Manager, not the ANSI driver, must convert SQL_C_WCHAR (Unicode) data to SQL_CHAR (ANSI) data, and vice versa (see Unicode Support in ODBC for a detailed discussion). This is necessary because ANSI drivers do not support any Unicode ODBC types.

The Driver Manager must use client code page information (Active Code Page on Windows and the IANAAppCodePage attribute on UNIX/Linux) to determine which ANSI code page to use for the conversions. The Active Code Page or IANAAppCodePage must match the database default character encoding. If not, conversion errors are possible.

Default Unicode Mapping

The default Unicode mapping for an application's SQL_C_WCHAR variable is:

Platform	Default Unicode Mapping
Windows	UCS-2/UTF-16
AIX	UTF-8
HP-UX	UTF-8
Solaris	UTF-8
Linux	UTF-8

Connection Attribute for Unicode

If you do not want to use the default Unicode mappings for SQL_C_WCHAR, a connection attribute is available to override the default mappings. This attribute determines how character data is converted and presented to an application and the database.

SQL_ATTR_APP_WCHAR_TYPE (1061): Sets the SQL_C_WCHAR type for parameter and column binding to the Unicode type, either SQL_DD_CP_UTF16 (default for Windows) or SQL_DD_CP_UTF8 (default for UNIX/Linux)..

You can set this attribute before or after you connect. After this attribute is set, all conversions are made based on the character set specified.

For example:

```
rc = SQLSetConnectAttr (hdbc, SQL_ATTR_APP_WCHAR_TYPE,
(void *)SQL_DD_CP_UTF16, SQL_IS_INTEGER);
```

SQLGetConnectAttr and SQLSetConnectAttr for the SQL_ATTR_APP_WCHAR_TYPE attribute return a SQL State of HYC00 for drivers that do not support Unicode.

This connection attribute and its valid values can be found in the file qesqlext.h, which is installed with the product.

NOTE: For the SQL Server Wire Protocol driver, this attribute is supported only on UNIX and Linux, not on Windows.

The Driver Manager and Unicode Encoding on UNIX

Unicode ODBC drivers on UNIX and Linux can use UTF-8 or UTF-16 encoding. This would normally mean that a UTF-8 application could not work with a UTF-16 driver, and, conversely, that a UTF-16 application could not work with a UTF-8 driver. To accomplish the goal of being able to use a single UTF-8 or UTF-16 application with either a UTF-8 or UTF-16 driver, the Driver Manager must be able to determine with which type of encoding the application and driver use and, if necessary, convert them accordingly.

To make this determination, the Driver Manager supports two ODBC environment attributes: SQL_ATTR_APP_UNICODE_TYPE and SQL_ATTR_DRIVER_UNICODE_TYPE, each with possible values of SQL_DD_CP_UTF8 and SQL_DD_CP_UTF16. The default value is SQL_DD_CP_UTF8.

The Driver Manager undertakes the following steps before actually connecting to the driver:

1. Determine the application Unicode type: Applications that use UTF-16 encoding for their string types need to set SQL_ATTR_APP_UNICODE_TYPE accordingly before connecting to any driver. When the Driver Manager reads this attribute, it expects all string arguments to the ODBC "W" functions to be in the specified Unicode format. This attribute also indicates how the SQL_C_WCHAR buffers must be encoded.
2. Determine the driver Unicode type: The Driver Manager must determine through which Unicode encoding the driver supports its "W" functions. This is done as follows:
    - SQLGetEnvAttr(SQL_ATTR_DRIVER_UNICODE_TYPE) is called in the driver by the Driver Manager. The driver, if capable, returns either SQL_DD_CP_UTF16 or SQL_DD_CP_UTF8 to indicate to the Driver Manager which encoding it expects.

- If the preceding call to SQLGetEnvAttr fails, the Driver Manager looks either in the Data Source section of the odbc.ini specified by the connection string or in the connection string itself for a connection option named DriverUnicodeType. Valid values for this option are 1 (UTF-16) or 2 (UTF-8). The Driver Manager assumes that the Unicode encoding of the driver corresponds to the value specified.

- If neither of the preceding attempts are successful, the Driver Manager assumes that the Unicode encoding of the driver is UTF-8.

3. Determine if the driver supports SQL_ATTR_WCHAR_TYPE: SQLSetConnectAttr (SQL_ATTR_WCHAR_TYPE, x) is called in the driver by the Driver Manager, where x is either SQL_DD_CP_UTF8 or SQL_DD_CP_UTF16, depending on the value of the SQL_ATTR_APP_UNICODE_TYPE environment setting. If the driver returns any error on this call to SQLSetConnectAttr, the Driver Manager assumes that the driver does not support this connection attribute.

   If an error occurs, the Driver Manager returns a warning. The Driver Manager does not convert all bound parameter data from the application Unicode type to the driver Unicode type specified by SQL_ATTR_DRIVER_UNICODE_TYPE. Neither does it convert all data bound as SQL_C_WCHAR to the application Unicode type specified by SQL_ATTR_APP_UNICODE_TYPE.

Based on the information it has gathered prior to connection, the Driver Manager either does not have to convert function calls, or, before calling the driver, it converts to either UTF-8 or UTF-16 all string arguments to calls to the ODBC "W" functions.

UTF-16 Applications on UNIX

Because the DataDirect Driver Manager allows applications to use either UTF-8 or UTF-16 Unicode encoding, applications written in UTF-16 for Windows platforms can also be used on UNIX and Linux platforms.

The Driver Manager assumes a default of UTF-8 applications; therefore, two things must occur for it to determine that the application is UTF-16:

- The definition of SQLWCHAR in the ODBC header files must be switched from "char *" to "short *." To do this, the application uses #define SQLWCHARSHORT.

- The application must set the ODBC environment attribute SQL_ATTR_APP_UNICODE_TYPE to a value of SQL_DD_CP_UTF16, for example:

```
rc = SQLSetEnvAttr(*henv, SQL_ATTR_APP_UNICODE_TYPE,
(SQLPOINTER)SQL_DD_CP_UTF16, SQL_IS_INTEGER);
```

Summary

Although Unicode was developed to expand the number of available characters and ultimately to simplify data access in a world-wide setting, these goals have not been fully realized. The character set has been expanded, but data access still involves a number of conversions. This is because Unicode must be able to work with existing ANSI applications and because database vendors make data available in a number of different Unicode encoding formats, including UCS-2, UTF-16, and UTF-8.

ODBC drivers and the ODBC Driver Manager are the components responsible for processing function call and data encoding conversions. Developers of these components must code them to be able to recognize the type of function call and the various Unicode encoding schemes, and to make the appropriate conversions. The drivers and Driver Manager must make these conversions; Unicode data in a database can be accessed only by W function calls, and ANSI data can only be accessed by standard, non-W function calls.

Application developers, on the other hand, need only consider whether a Unicode or ANSI application is most appropriate for a particular circumstance and code their function calls appropriately—W function calls, such as SQLConnectW, for Unicode, or standard function calls, such as SQLConnect, for ANSI. They can also code an application to switch dynamically between Unicode and ANSI calls.

As Unicode applications and data become more prevalent, and more agreements are reached concerning encoding and implementation of Unicode, data access will become more efficient as the need for function call and data conversion is reduced.

We welcome your feedback! Please send any comments concerning documentation, including suggestions for other topics that you would like to see, to:

docgroup@datadirect.com

FOR MORE INFORMATION

800-876-3101

Worldwide Sales

Belgium (French).....0800 12 045
Belgium (Dutch).....0800 12 046
France.....0800 911 454
Germany 0800 181 78 76
Japan 0120.20.9613
Netherlands.....0800 022 0524
United Kingdom.....0800 169 19 07
United States.....800 876 3101

DataDirect Technologies is focused on standards-based data connectivity, enabling software developers to quickly develop and deploy business applications across all major databases and platforms. DataDirect Technologies offers the most comprehensive, proven line of data connectivity components available anywhere. Developers worldwide at more than 250 leading independent software vendors and thousands of corporate IT departments rely on DataDirect® products to connect their applications to an unparalleled range of data sources using standards-based interfaces such as ODBC, JDBC™ and ADO.NET. Developers also depend on DataDirect to radically simplify complex data integration projects using XML products based on the emerging XQuery and XQJ standards. DataDirect Technologies is an operating company of Progress Software Corporation (Nasdaq: PRGS), a US$300+ million global software industry leader. Headquartered in Bedford, Mass., DataDirect Technologies can be reached on the Web at http://www.datadirect.com or by phone at +1-800-876-3101.