

SYBASE®

**BUILDING INTERNATIONALIZED APPLICATIONS
WITH SYBASE POWERBUILDER®**

THE
ENTERPRISE.
UNWIRED.



TABLE OF CONTENTS

Introduction	3
Internationalization and Localization Basic Terminology	3
Internationalizing PowerBuilder Applications	3
Localizing PowerBuilder Applications	4
What is Unicode?	5
Byte-Order Mark	6
PowerBuilder 10 and Unicode	6
PowerBuilder 10 Support for DBCS and Unicode Databases	6
ASE Unicode Support and PowerBuilder	8
Oracle Unicode Support and PowerBuilder	8
ODBC/ JDBC /OleDB/ ADO.Net Database Support	8
Informix Native Support and PowerBuilder 10	8
PowerScript Support for Unicode in PowerBuilder 10	8
Using PowerScript to Manipulate ANSI & Unicode Strings	8
PowerScript String Manipulation Functions	9
PowerScript Functions for Processing Unicode Files	10
DataWindow and Web DataWindow	10
Web DataWindow	11
JSP Authoring & Web Services	11
PowerBuilder Native Interface and ORCA APIs	11
External Function Calls	12
Applications that Meet the Needs of Globally Dispersed Users	12

INTRODUCTION

Sybase PowerBuilder 10 supports a comprehensive set of features for working with Unicode and building multi-language, internationalized applications. The multi-language features are implemented at a fundamental level; the PowerBuilder environment itself is now converted to support Unicode. The Unicode-enabled underlying system drives Unicode-enabled PowerBuilder applications that are capable of reading files and databases containing multi-language content and presenting this content to end users in a culturally appropriate rendering of their language. This whitepaper discusses topics related to Unicode-enabling PowerBuilder applications from a developer perspective; it also includes background on Unicode and user interface design as it relates to multi-language support.

There was a time in the not so distant past when the vast majority of developers never had to think about multi-language applications. The tiny minority of programmers who did need to worry about internationalized code usually worked for an organization with branch offices in another country where the remote branch's sales manager was a top producer with high visibility accounts and lots of pull. Over time, some of those programmers took their experience and developed a tidy specialty career as internationalization experts because, in those not too distant days, making applications multi-language capable was a daunting task. That was in the past. Nowadays, the global economy is becoming a global reality. The advent of the Internet thrust accessible technology into the worldwide consciousness at an accelerated pace. Internationally, computer literacy soared as computers became affordable and the Web made them truly useful; the different language packs for Microsoft® Windows® became extremely popular and the CDs stopped being used as drink coasters. The tools and infrastructure for supporting applications in multiple markets bloomed and internationalized application development moved from the realm of specialists into the sphere of mainstream developers.

INTERNATIONALIZATION AND LOCALIZATION BASIC TERMINOLOGY

Converting an existing PowerBuilder application to multiple languages requires two main passes through the code and user interfaces, an internationalization pass to modify the code and a localization pass for each target language.

The first pass, *internationalization*—also known as I18N, meaning an ‘I’ followed by 18 characters and an ‘N’—is the process of enhancing an application to work with multiple languages and their attendant character sets; this means making the code culturally independent and able to process multiple languages simultaneously. The PowerBuilder 10 Migration Assistant acts as a guide and identifies the necessary changes for Unicode-enabling legacy applications.

The next conversion pass is *localization*. Localization is performed for each supported language; it includes translating all strings the user sees and verifying the control sizing and arrangement. The width of displayed text can vary greatly between languages, so a string displayed in English that fits nicely in a button control may be too long when displayed in Russian. A general rule while designing or internationalizing the application is to size text-based controls at 1.3 times larger than the space needed to hold an English string; this will handle most real estate differences in the size of displayed strings rendered into multiple languages. Some languages, like Hebrew and Arabic, are displayed from right to left; this can affect the choice of left or right justification of text displayed in controls.

INTERNATIONALIZING POWERBUILDER APPLICATIONS

Sybase PowerBuilder 10 is Unicode-enabled and supports creating multi-language capable applications right out of the box. For existing PowerBuilder applications, internationalization is usually an intelligent enhancement; it broadens potential markets for applications, it makes the application more nimble and able to quickly address previously unforeseen international market opportunities, and it extends the lifespan of existing code. With Unicode, internationalization is straightforward and PowerBuilder 10 does most of the work with the Migration Assistant.

Internationalization – In terms of the internationalization effort, the process of moving an existing application to Sybase PowerBuilder 10 takes care of most issues. Strings are transparently multi-language enabled and string handling functions perform the same operations, only in a multi-language context. In the rare cases where strings need to be handled in a byte-by-byte fashion, PowerBuilder 10 has added a new set of byte-oriented string handling functions. The Migration Assistant performs common internationalization changes and identifies areas needing further developer attention.

Most of the database interfaces have been modified to support the exchange of multi-byte and Unicode information. File handling has been extended to identify specific Unicode formats of existing files and read and write in the target format. PowerBuilder's relatively painless migration is a natural consequence of PowerBuilder's architecture. PowerBuilder insulates application developers from most underlying native system issues, so between the inherent architecture and the Migration Assistant, developers reap the benefits of internationalized applications—without paying a hefty price in development effort.

LOCALIZING POWERBUILDER APPLICATIONS

Once the internationalization pass is complete, the next phase is localizing the PowerBuilder application into the targeted languages. Localization has three basic steps:

- Extract all text from the application's PowerBuilder libraries
- Translate the extracted text into the target languages
- Apply the translated text back to individual localized PowerBuilder libraries

Translation Tool Kit – The Translation Toolkit is a set of Sybase tools for localizing PowerBuilder applications. It supplies localized PowerBuilder runtime files for French, German, Italian, Spanish, Dutch, Danish, Norwegian, and Swedish. These runtime files can be installed in the development environment and on the end user's machine. The localized runtime files handle language-specific data and are required to display—in the local language—standard dialog boxes and user interface elements, such as day and month names in spin controls. The localized runtimes also contain translated PowerBuilder error messages. The Translation Toolkit includes the dictionaries used to make the localized PowerBuilder runtime file translations. Developers can use the dictionaries to help translate application-specific strings into the target language and achieve a higher degree of consistency with the translation.

Fully testing a localized application requires a localized operating system for the target language. The localized operating system provides references to System objects, such as icons and buttons referenced using enumerated types in PowerBuilder, like OKCancel!, YesNo!, Information!, and Error!. These enumerated types rely on API calls to the local operating system, which passes back the appropriately localized button, icon or symbol for the target language.

Some development shops add a last step of hiring a contractor who is either a native speaker or highly literate in the targeted language. Taking this last step usually depends on the localized application's audience, the number of users, and the size of the development budget. When taking this step it is important to not only find someone with a high degree of literacy in the language, but also has an understanding of common user interface idioms used in the language. Anyone who has tried to follow a poorly translated set of instructions for a product understands that translation is much more than a one-to-one word substitution process.

WHAT IS UNICODE?

Unicode is an international standard that represents a major step forward in character set encoding. Unicode defines codes for all characters used in all major languages today with room for expansion. Unicode supports mixing any languages in the same text, because all the characters needed are represented in the same, universal encoding. Strings containing Unicode are easily transferred between dissimilar systems without corruption.

By the measure of such a basic unit as a character set, the Unicode standard is fairly new, at least as compared to something like ASCII that was first standardized in 1963. While ASCII was an important standard for its time, the single-byte ASCII approach quickly demonstrated limitations for non-English languages, even when using language specific code pages; it proved itself woefully inadequate for rendering the thousands of glyphs used in Asian character sets.

The advent of Unicode neatly solved the character-mapping problem by allocating a unique value for every known character. Oddly enough, Unicode adoption moved at a relatively slow pace until the Web put the world online and computers from across the planet started to exchange text in different languages, with often ridiculously garbled results. This necessitated a harder look at an all-encompassing solution. Unicode proved to be an answer that was ready and relatively easily implemented.

Unicode provides a unique number for every character, no matter what the platform, no matter what the program—be it a Web, client/server, or standalone application—no matter what the language. The beauty of Unicode is the complete lack of ambiguity, as long as the Unicode Transformation Format (UTF) is known, any character in any language can be handled and displayed. This is an extremely powerful tool for managing text and is transparently supported by PowerBuilder 10 applications.

Unicode characters are identified with 'U+' followed by a hexadecimal number. For example, U+05E9 is the Hebrew letter Shin (ש).

- The Unicode Standard contains most common characters in the Basic Multilingual Plane (BMP), the first 65,535 characters in the standard. Rarer and historic characters are encoded on the supplementary planes, and have Unicode values from U+10000 to U+10FFFF.

The three common Unicode Transformation Formats (UTF) are:

- **UTF-8:** Uses 1 to 4 bytes to represent each Unicode character. A nice aspect of UTF-8 is that ASCII characters match the UTF-8 characters; in other words, the lower end of the UTF-8 codes 0 through 127 is the same as those in the ASCII character set. One (1) to 3 bytes are needed in UTF-8 to represent Unicode characters in the BMP. Four (4) bytes are required to represent Unicode characters in the supplementary planes.
- **UTF-16:** Uses one 16-bit unit to represent each Unicode character in the BMP, and two 16-bit units to represent Unicode characters in the supplementary planes.
- **UTF-32:** Use one 32bit unit to represent each Unicode character.

In Web pages, encoding information is passed in HTML using the meta tag:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Encoding information is passed in email with the following text in the header:

```
Content-Type: text/plain; charset="UTF-8"
```

The Windows Character Map tool can be useful for looking at Unicode characters. To actually render the character, a font with a mapping to the character needs to be on the machine. The following are examples of Unicode letters copied using the Character Map tool:

(**Ж**) – U+0416 Cyrillic capital letter Zhe

(**ف**) – U+0641 Arabic letter Feh

(**d**) – U+0064 Latin small letter D

BYTE-ORDER MARK

Unicode plain txt files may prefix a byte-order mark at the beginning of the text; in reality, not all Unicode files contain this mark. The byte-order mark is used to identify the placement of the least significant byte. Intel and MIPS systems use little endian ordering—meaning the least significant byte comes first—and Motorola processors use the big endian ordering, which positions the least significant byte last. By default, PowerBuilder 10 and Microsoft use the UTF-16 little endian form. The byte-order mark does not resolve to any Unicode character and is not displayed in character rendering. PowerScript functions that write Unicode text to files and the Blob type use an encoding specification in the function calls; the function calls automatically prepend the byte-order information to the text. The following list shows the byte-order mark for different Unicode formats:

EF BB BF	UTF-8
FE FF	UTF-16, little endian (PowerBuilder 10 default)
FF FE	UTF-16, big endian
FF FE 00 00	UTF-32, little endian
00 00 FE FF	UTF-32, big-endian

POWERBUILDER 10 AND UNICODE

PowerBuilder 10 uses Unicode (UTF-16 little endian) internally. PowerBuilder is written in C++ and all internal string manipulation is Unicode compliant as of PowerBuilder 10. PowerBuilder versions 7, 8, and 9 used double-byte character sets (DBCS); and PowerBuilder versions 6 and earlier used ASCII. This internal upgrade to Unicode means all PowerBuilder 10 applications are also fully capable of supporting Unicode and therefore multiple languages. PowerBuilder 10 has added PowerScript functions to process Unicode and ANSI/DBCS characters and the sources for the characters, such as Unicode and ANSI/DBCS databases and files.

POWERBUILDER 10 SUPPORT FOR DBCS AND UNICODE DATABASES

PowerBuilder's specialty is providing reading and editing views into serious information storage. Increasingly, this storage contains multi-lingual information. Because a PowerBuilder application is much more than a series of windows, an obvious companion to PowerBuilder's ability to manipulate and render internationalized information is a data source capable of supplying and consuming this information. PowerBuilder 10 supports an impressive array of database interfaces that support Unicode and ANSI/DBCS characters.

The following table lists the PowerBuilder 10 database interfaces and their support for international character sets:

	Client/Server		N-tier	
	ANSI DB	Unicode DB / Unicode Cols	ANSI DB	Unicode DB
ASE 12/12.5	–	–	–	–
SYC	Yes	Yes	–	–
SYJ	–	–	Yes	Yes
ODBC for ASE	Yes	Yes	Yes	Yes
JDBC for ASE	Yes	Yes	Yes	Yes
OleDB for ASE	Yes	Yes	Yes (Win)	Yes (Win)
ASA 8/9	–	–	–	–
ASA ODBC	Yes	Yes	Yes	Yes
ASA JDBC/OleDB	Yes	Yes	Yes	Yes
Oracle 8/9	–	–	–	–
O90	Yes	Yes	Yes	Yes
O84	Yes	Yes	Yes	Yes
ODBC for Oracle	Yes	Yes	Yes	Yes
JDBC for Oracle	Yes	Yes	Yes	Yes
OleDB for Oracle	Yes	Yes	Yes (Win)	Yes (Win)
MSS	–	–	–	–
OleDB	Yes	Yes	Yes (Win)	Yes (Win)
ODBC/JDBC	Yes	Yes	Yes	Yes
DB2	–	–	–	–
ODBC	Yes	Yes	Yes	Yes
Informix	–	–	–	–
Native	Yes	No	Yes	No
ODBC	Yes	Yes	Yes	Yes

ASE UNICODE SUPPORT AND POWERBUILDER

For Unicode support, the Sybase Adaptive Server Enterprise (SYC/SYJ) database Interface has a new “UTF8” database parameter. To enable Unicode for Multilanguage in the database driver, select or create a DB profile and go to the profile’s Properties >> Regional Settings and select the checkbox “UTF8 character Set Installed or Unicode Conversions in Server”. This sets the UTF8 Database parameter to 1; its default is 0.

There are two steps in Unicode-enabling PowerBuilder with ASE. First, the UTF8 database parameter must be set to 1, *and* secondly, the server must be specifically configured to support both Adaptive Server direct conversions and Unicode conversions. To configure Unicode conversion the database administrator must run the following command on the server:

```
sp_configure, "enable Unicode conversion", 1
```

For more information on ASE’s support for Unicode and its datatypes, visit the online manual at <http://www.sybase.com/support/manuals>

ORACLE UNICODE SUPPORT AND POWERBUILDER

For Client/Server PowerBuilder 10 applications, the O90/O84 Oracle database interface for PowerBuilder 10 can exchange ANSI and Unicode without special settings.

N-tier PowerBuilder 10 applications using the O90 Oracle database interface connecting to Oracle 9 databases using the connection cache need an EAServer patch. The patch for EAServer 4.2.3/5.1 adds a new, predefined connection cache “Oracle9U”; this connection handle gives PowerBuilder components access to Oracle Unicode data.

ODBC/ JDBC /OLEDB/ ADO.NET DATABASE SUPPORT

Client/Server PowerBuilder 10 applications can exchange Unicode and ANSI data with databases supported by the ODBC, JDBC, OleDb, and ADO.NET database interfaces without requiring special settings.

N-tier PowerBuilder 10 applications using ODBC connecting to Adaptive Server Anywhere Unicode databases through the connection cache will need a special patch for EAServer 4.2.3/5.1. The patch adds a new predefined connection type called “EASDemoU”. This “EASDemoU” connection type is required to access Unicode data from a Unicode ASA database.

INFORMIX NATIVE SUPPORT AND POWERBUILDER 10

PB10 can consume ANSI/DBCS data from the Informix database without any special settings.

The Informix native driver does not currently support access to Unicode databases so the Informix Unicode database is not supported in PB10.

POWERSCRIPT SUPPORT FOR UNICODE IN POWERBUILDER 10

PowerBuilder 10 has additional PowerScript functions and data types for processing Unicode strings and ANSI/DBCS strings. This includes file handling functions for manipulating ANSI/DBCS and Unicode files.

USING POWERSCRIPT TO MANIPULATE ANSI & UNICODE STRINGS

String Type – The string type will always be a Unicode string. PowerBuilder 10 processes all strings as Unicode so all data in a string will be Unicode; there are no ANSI strings any more. Since all Unicode characters are unique, regardless of the language, a string can contain a mix of multiple language characters. While this is possible, in practice this is rarely a real-world scenario for an application since a user will want any text presented to them to be in their native language.

Blob Type – Blob remains as a binary data type. A Blob can store binary data, ANSI characters, or Unicode characters.

The String() and Blob() functions support the conversions between the types. The functionality contained in the String() and Blob() functions obviate the need for the older functions FromANSI(), ToANSI(), FromUnicode(), and ToUnicode(). These functions are still supported in PowerBuilder 10, but deprecated; users are encouraged to migrate to the String() and Blob() functions.

The String() and Blob() functions require an encoding parameter. The encoding parameter is one of the following:

- EncodingANSI!
- Encoding UTF8!
- EncodingUTF16LE! – UTF-16 Little Endian encoding (PowerBuilder 10 default)
- EncodingUTF16BE! – UTF-16 Big Endian encoding

To convert a Blob to a String use:

```
String ( blob, {Encoding} )
```

For example:

```
String str
```

```
str = String(Blb) // Absence of encoding parameter defaults to UTF16LE
```

or

```
String str
```

```
str = String(Blb, EncodingUTF8!) // Use UTF8 encoding
```

To convert a String to a Blob use:

```
Blob ( string, {Encoding} )
```

For example:

```
Blob Blb
```

```
Blb = Blob("Some Text") // Absence of encoding parameter defaults to UTF16LE
```

or

```
Blob Blb
```

```
Blb = Blob("Some Text", EncodingUTF8!) // Use UTF8 encoding
```

POWERSCRIPT STRING MANIPULATION FUNCTIONS

In PowerBuilder 10, the PowerScript functions Len(), Left(), Mid(), and Right() are all Unicode character based and are equivalent to the existing LenW(), LeftW(), MidW(), and RightW() functions.

```
len("ABC-这是一个测试") ==> 10  
len("これはテストである") ==> 9
```

Unicode Len() function example

To manipulate strings as bytes or ASCII characters instead of Unicode characters, a new set of LenA(), LeftA(), MidA(), and RightA() functions have been added. When the 'A' functions are applied, PowerBuilder will convert the Unicode string to a DBCS string based on the machine's locale, and then apply the operation.

```
lenA("ABC-这是一个测试") ==> 16
```

Unicode LenA() function example

POWERSCRIPT FUNCTIONS FOR PROCESSING UNICODE FILES

PowerBuilder 10 adds PowerScript support for Unicode files. The FileOpen() function now has an Encoding argument for identifying the encoding of the target file. The function FileEncoding(filename) determines and returns the encoding (EncodingANSI!, Encoding UTF8!, EncodingUTF16LE!, And EncodingUTF16BE!) used in the file. The FileEncoding() function should precede the opening of existing files so the correct encoding parameter can be passed in the FileOpen() call.

When reading from or writing to a file, the conversion to the encoding scheme will occur automatically, according to the encoding mode used to open the file.

```
// Read an Ansi File

Integer li_FileNum

String s_rec

li_FileNum = FileOpen("Employee.txt", TextMode!, Read!,
EncodingANSI!)

//li_FileNum = FileOpen("Employee.txt", TextMode!)

FileRead(li_FileNum, s_rec)

// Read a Unicode File

Integer li_FileNum

String s_rec

li_FileNum = FileOpen("EmployeeU.txt", TextMode!, Read!,
EncodingUTF16LE!)

FileRead(li_FileNum, s_rec)

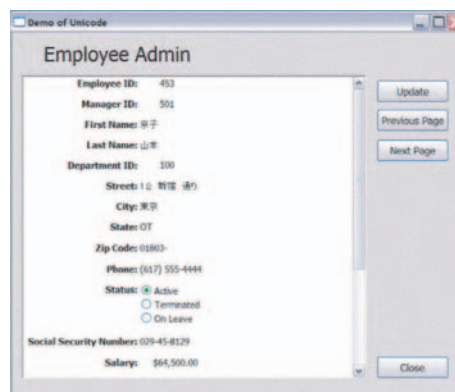
// Read a Binary File

Integer li_FileNum

blob bal_rec

li_FileNum = FileOpen("Employee.imp", Stream Mode!, Read!)
```

DATAWINDOW® AND WEB DATAWINDOW



The DataWindow is a central feature of most PowerBuilder applications. In PowerBuilder 10, the DataWindow and Web DataWindow fully support multi-language display and manipulation for Select, Insert, and Update with the supported databases. The string and file manipulation functions are consistent with the PowerScript functions for handling Unicode and ANSI multi-byte strings.

WEB DATAWINDOW

Static HTML Pages – For Web DataWindows with data sources that may contain Unicode characters, the generated Web page needs to reflect the specific character set instead of the default iso-8859-1 character set. To do this, set the <META> tag in the <HEAD> section of generated pages. The following HTML code configures the page as UTF-8 encoding:

```
<META content="text/html; charset=utf-8" http-equiv="Content-Type">
```

EAServer – When working with EAServer, setting the `com.sybase.jaguar.component.code.set` property for each component can change individual component's character set. By default, a component uses the server's codeset.

ODBC Connections – For Web DataWindows that use ODBC in code, use the following command to set the database parameter of the profile:

```
ODBCU_CONLIB = 1
```

JSP AUTHORIZING & WEB SERVICES

Java Server Pages (JSP) are fully capable of supporting Unicode and ANSI requests. JSP files are saved in UTF-8 format. Additionally, PowerBuilder 10 has Unicode-enabled the JSP authoring tool.

JSP Pages – For JSP pages to display Unicode, the character set for the response object in a page directive needs to be set:

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

JSP Server-Side Request Object – The JSP request object needs an assigned character set. This is done with a server-side call to `setCharacterEncoding()`. The following example sets the character set for the request object to UTF-8:

```
request.setCharacterEncoding("UTF8");
```

Web services – In PB10, the Web services client handles international characters.

POWERBUILDER NATIVE INTERFACE AND ORCA APIS

PowerBuilder Native Interface – With PowerBuilder 10, the PBNI supports two application programming interfaces, an ANSI/DBCS interface and a Unicode interface. Using a chosen interface, developers can create PowerBuilder extensions to support their chosen character set. The PBNI has templates to guide users in developing extensions.

ORCA – ORCA is the C interface for accessing the PowerBuilder Library Manager functions used by PowerBuilder in the Library Painter. With PowerBuilder 10, ORCA offers both an ANSI/BDCS interface and a Unicode interface.

EXTERNAL FUNCTION CALLS

PowerBuilder has long supported making external function calls by mapping local or global functions that call out to the external system or 3rd party DLLs. To support Unicode, the syntax of external function calls need to change slightly.

In PowerBuilder 9 and before, the syntax was:

```
FUNCTION int MessageBoxA(int handle, string content, string title,  
int show type) LIBRARY "user32.dll"
```

In PowerBuilder 10, the syntax is now:

```
// Use ANSI version of system function or 3rd party Dlls  
  
FUNCTION int MessageBox(int handle, string content, string title,  
int showtype) LIBRARY "user32.dll" ALIAS FOR "MessageBoxA;ansi"  
  
  
// Use Unicode version of system function or 3rd party Dlls  
  
FUNCTION int MessageBox(int handle, string content, string title,  
int showtype) LIBRARY "user32.dll" ALIAS FOR "MessageBoxW"
```

APPLICATIONS THAT MEET THE NEEDS OF GLOBALLY DISPERSED USERS

Sybase PowerBuilder 10 is a complete multi-language application development system. It supports Unicode and ANSI string, file, and database manipulation for all interfaces, including the DataWindow. JSP and Web-based deployment of PowerBuilder technology also supports multiple languages. The Migration Assistant and Translation Toolkit guide developers as they create multi-language applications or retrofit legacy applications to be multi-language capable.

The PowerBuilder development environment continues to evolve as it matches the latest trends in application development, including I18N and mobile platforms. Sybase has an ongoing commitment to PowerBuilder and continues to enhance PowerBuilder's capabilities. PowerBuilder's basic architecture has proved itself over multiple releases and feature set upgrades to be a robust, rapid application development platform.

PowerBuilder is a development platform, not a language. Unlike the majority of applications that rely on a development language, PowerBuilder applications reap immediate benefits when changes are made to the underlying PowerBuilder development platform. Using PowerBuilder, applications are rapidly developed, easily maintained, and are continuously brought up to the ever-changing state of the art by new releases. Seamless multi-language support demonstrates that, once again, the PowerBuilder architecture was, and is, the right choice for data-intensive application development.

SYBASE®

Sybase Incorporated
Worldwide Headquarters
One Sybase Drive
Dublin CA, 94568 USA
T 1.800.8.SYBASE
www.sybase.com

Copyright © 2005 Sybase, Inc. All rights reserved. Unpublished rights reserved under U.S. copyright laws. Sybase, the Sybase logo, DataWindow and PowerBuilder are registered trademarks of Sybase, Inc. All other trademarks are property of their respective owners. ® indicates registration in the United States. Printed in the U.S.A. L02684 5/05