Omnis for Unicode

White Paper

August 2009

Contents

Omnis for Unicode

Introduction: Unicode or non-Unicode?

Did you know the next major release of Omnis Studio will be Unicode only? At the moment, in Omnis Studio 4.3.x, we ship the Unicode and non-Unicode versions of Omnis and let developers decide which version to use. The Unicode version of Omnis was released with Studio 4.1 in 2005 and is being used by several developers who need to store data containing extended characters, usually to support languages other than English. All that the switch to the Unicode-only Omnis.exe means is that it will support the full array of foreign and extended character sets available on modern computer platforms and the internet, nothing else will change. It is a necessary change to support modern computing platforms and associated software products, like server databases and third-party programming languages, many of which integrate with or are supported in Omnis.

This white paper summarises all the issues regarding the switch to Unicode only, and describes some of the technical detail - none of the issues are that huge, and in fact switching to Unicode will bring many advantages, not least it could open up your application to new international markets that require data to be stored in extended characters. Even if your market does not require a switch to Unicode, Omnis will become Unicode-only so you may benefit from knowing about the issues regarding Unicode.

Existing Omnis developers should note that much of the information in this white paper has been published before, but is brought together here and updated for your convenience.

What is Unicode?

According to the Unicode Consortium (www.unicode.org), who maintains the Unicode standard:

> *"Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language."*

A fuller explanation by the Unicode Consortium is at:
http://www.unicode.org/standard/WhatIsUnicode.html

To further quote from the Unicode Consortium:

> *"Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers. No single encoding could contain enough characters... Incorporating Unicode into client-server or multi-tiered applications and websites offers significant cost savings over the use of legacy character sets. Unicode enables a single software product or a single website to be targeted across multiple platforms, languages and countries without re-engineering. It allows data to be transported through many different systems without corruption." (www.unicode.org, July 2009)*

And that's why it's important that Omnis, and more importantly *your applications*, should provide support for Unicode data going forward. From a commercial perspective, you ought to provide support for Unicode in your applications to at least *maintain* your current position in the marketplace (otherwise you may lose out to your competitors), and to *gain new customers* in new and emerging markets that require support for data with foreign and extended characters.

The Unicode Consortium provides information and resources concerning Unicode, including the standard definition and maintenance, character code tables, a locale identifier repository, and lists of Unicode enabled products.

The latest version of Unicode is version 5.1 which is capable of representing over 100,000 different characters, used in many different languages throughout the world. Many operating systems and

software products support Unicode by default, which is now universally accepted as the standard for character representation. For example, Microsoft Windows NT, 2000, XP, and Vista, as well as the latest Mac OS X all offer Unicode support. All new web standards, such as the latest versions of HTML and XML support Unicode, as well as the latest versions of Internet Explorer and all Mozilla-based browsers. In addition, the most recent versions of Sybase, Oracle, and DB2 all provide Unicode support.

Unicode support in Omnis

Omnis Studio 4.1 was the first version to support Unicode. When this version was released there was a separate Unicode compatible version of Omnis Studio, together with the non-Unicode version of Omnis. At present, the Unicode version is available for Windows and Mac OS X only. The current shipping version of Omnis Studio 4.3.1 also provides a Unicode and non-Unicode of Omnis, but with the next major release of Omnis, we will supply only the Unicode version of Omnis Studio, adding Linux support to the existing Windows and Mac OS X versions.

Together with the display of text strings and character data in multiple languages within the Omnis development and runtime environments, the use of Unicode encoding primarily affects the sort order of dynamic data, in lists for example, as well as the querying and retrieval of data from Unicode compatible Server databases and the Omnis database itself.

Character Normalization

Originally, Unicode was a 16-bit character set. It has subsequently been extended to include code point values up to and including U+10FFFF. It is not expected that it will be extended any further. Windows and Mac OS X still represent Unicode character strings using arrays of Short (16-bit) integers. This is not a problem, because the UTF-16 standard allows code points U+10000 and greater to be represented by pairs of 16-bit values (each member of the pair occupies space in the 16-bit range that is not used for code points). This representation is referred to as a *surrogate pair*.

Internally Omnis uses UTF-32 to represent code points, that is, each code point occupies 32 bits, and the value of each code point is between 0 and U+10FFFF inclusive. This allows for straightforward processing of character strings, since every code point occupies the same space in memory.

Unicode allows a significant number of characters to be represented by more than one sequence of code points. For example, consider the letter E with circumflex and dot below (Ệ), a character that occurs in Vietnamese. This character has five possible representations in Unicode:

1. U+0045 Latin capital letter E
   U+0302 combining circumflex accent
   U+0323 combining dot below
2. U+0045 Latin capital letter E
   U+0323 combining dot below
   U+0302 combining circumflex accent
3. U+00CA Latin capital letter E with circumflex
   U+0323 combining dot below
4. U+1EB8 Latin capital letter E with dot below
   U+0302 combining circumflex accent
5. U+1EC6 Latin capital letter E with circumflex and dot below

A character represented by more than one code point is referred to as a *composite character*. A character represented by a single code point is referred to as a *pre-composed character*.

As far as the end-user is concerned each of these representations usually needs to be treated identically. This leads to some interesting consequences for Omnis. These are discussed in the following sections. Note the term *end-user character* means the character that the end-user is working with – in the example above, the end-user character is Ệ.

Normalization of a Unicode character string converts the string into a standard, defined format. Once normalized, a Unicode character string has only one possible representation, thereby making it possible to compare it with other character strings, and produce results useful to the end-user. The Unicode standard recommends two forms of normalization. These are:

- **Canonical decomposition, referred to as NFD:**
  Pre-composed characters are replaced by their equivalent composite characters; Composite characters are replaced with a single fixed composite representation.
- Canonical decomposition followed by canonical composition, referred to as NFC:
  After carrying out NFD, all composite characters are replaced with their pre-composed equivalent, where one exists.

Omnis provides two functions to normalize character strings:

- nfd(string)
  carries out canonical decomposition on the string and returns the normalized string.
- nfc(string)
  carries out canonical decomposition followed by canonical composition on the string and returns the normalized string.

These functions are **not** available in client-side web client methods.

Comparing Text

Omnis uses two types of comparison for character strings:

- Comparison of the UTF-8 values of the strings. This is called Character comparison.
- Comparison according to the rules for the locale specified via the localisation data file; prior to comparison, the input data is normalized. This is called National comparison. National comparison is more likely to produce results that the end-user would expect. Note that upper casing used in conjunction with national comparison may not have an effect, since sometimes the rules for the locale ignore the case of the characters.

The natcmp() function uses national comparison. Note that natcmp() is **not** available in client-side web client methods.

Omnis compares text for many different reasons, and in many different places. Key areas are:

- Sorting lists
- Searching lists
- Manipulating data file indexes
- Expressions, for example the test on an if statement

Omnis supports two types of character variable – character and national.

Sorting Lists

When using the character type, Omnis uses character comparison.

When using the national type, Omnis uses national comparison.

Searching Lists

When using the character type, Omnis uses character comparison.

Searches that directly use a character column of national type use national comparison.

Other searches, for example searches using a calculation, will behave as if they are operating on normal character data. However, you can use natcmp() as part of the calculation, in order to use national comparison.

Manipulating Data File Indexes

Indexes for national fields use national comparison.

Expressions

To ensure the correct behaviour of expressions that test the value of character variables, you must either normalize their value first using nfc() or nfd(), or you must use the natcmp() function.

Drawing Text

Depending on the font and operating system you use, different representations of the same end-user character may not always draw in the same way. The same applies if you try to use strings that require surrogate pairs. Generally speaking, you will get the best results if you normalize the text using nfc(), as the issues generally occur with composite characters.

Entering Text

Wherever possible, you are recommended to use the nfc() normalization form for data that is to be edited. If composite characters are present in the data, multiple left or right arrow key presses are required to skip a composite character, and also clicking and selecting in the text will highlight an area which when copied to the clipboard might not exactly contain what appeared to be highlighted.

Omnis performs NFC normalization on character data pasted from the clipboard when running in the thick client (runtime); no normalization occurs when pasting characters into a remote form when using the web client.

Character translation

The following functions allow you to translate a specified character in a string to its Unicode value and to allow the reverse.

- *unicode*(string,position[,returnhex])
  returns the Unicode value of the character at the specified position in the string. The first position in string is 1. If Boolean returnhex is true (default false) it returns a hex string representing the value, of the form 'U+h'.
- *unichr*(num1[,num2]...)
  returns a string formed by concatenating the supplied Unicode character codes. Each code is either a number or a string of the form 'U+h',where h is 1-6 characters representing a hexadecimal value.

These functions are available in client-side methods as well as the thick client. (Such functions will generate an error if used in the non-Unicode version of Omnis, making any of your libraries developed in the Unicode version of Omnis incompatible with the non-Unicode version of Omnis.)

Locale Identifier

The *locale()* function returns the Locale Identifier (LCID) for the current machine/operating system. As well as the language of the machine, the Locale Identifier specifies the decimal, thousand and list separators, currency values, units of measurement, date formats, and character sort order. The Locale is specified at the operating system level and is in the form language_country, where *language* is the ISO639 language name, and *country* is the ISO3166 country name. For example, the Locale for the UK is “en_GB”. On Mac OS X, there may be other information, such as a script code, between the language and country (this is because OS X uses ICU locales).

Note that *locale()* does not work in methods executing in the web client under Mac OS X.

Test for Unicode version

The *isunicode()* function returns true if and only if the current code is executing in the Unicode version of Omnis Studio. *isunicode()* also works in client-side methods, and indicates if the web client supports Unicode.

Unicode Data Conversion

The *uniconv()* function allows you to translate Unicode character data from one type to another. The syntax is:

```
uniconv(srctype,src,dsttype,dst,bom,errtext)
```

Converts *src*, and stores the result in *dst*. It returns zero for success, or a non-zero error code together with error text in *errtext*. *Src* and *dst* are either binary or character variables, depending on the values of the *srctype* and *dsttype*.

srctype and **dsttype** are one of the kUniType... constants (see below).

Bom is Boolean: if true, **dst** has a Unicode Byte Order Marker (BOM) if relevant for the destination type.

The kUniType... constants are as follows:

kUniTypeAuto: The source encoding is automatically detected from the conversion source; possible encodings are identified by the remaining kUniType... constants (only allowed for the source type).

kUniTypeUTF8: The data is stored in a binary variable and contains Unicode character data encoded using UTF-8

kUniTypeUTF16BE: The data is stored in a binary variable and contains Unicode character data encoded using UTF-16BE

kUniTypeUTF16LE: The data is stored in a binary variable and contains Unicode character data encoded using UTF-16LE

kUniTypeUTF16: The data is stored in a binary variable and contains Unicode character data encoded using UTF-16LE if the machine is little-endian, or UTF-16BE if the machine is big-endian. Useful when writing cross-platform code that interacts with the OS.

kUniTypeUTF32BE: The data is stored in a binary variable and contains Unicode character data encoded using UTF-32BE

kUniTypeUTF32LE: The data is stored in a binary variable and contains Unicode character data encoded using UTF-32LE

kUniTypeUTF32: The data is stored in a binary variable and contains Unicode character data encoded using UTF-32LE if the machine is little-endian, or UTF-32BE if the machine is big-endian. Useful when writing cross-platform code that interacts with the OS.

kUniTypeNativeCharacters: The data is stored in a binary variable and contains a stream of bytes, where each byte is a character in the Latin 1 character set for the machine (Ansi on Windows, MacRoman on the Mac, ISO-8859-1 on Unix

kUniTypeCharacter: The data is stored in a character variable. Note – this constant has been moved since the last Unicode build, so you need to re-enter it in your code.

kUniTypeAnsiThai, kUniTypeAnsiCentralEuropean, kUniTypeAnsiCyrillic, kUniTypeAnsiLatin1, kUniTypeAnsiGreek, kUniTypeAnsiTurkish, kUniTypeAnsiHebrew, kUniTypeAnsiArabic, kUniTypeAnsiBaltic, and kUniTypeAnsiVietnamese: The data is stored in a binary variable, and contains character data where each byte is encoded using the identified ANSI code page.

Formfile

The $filereadencoding and $filewriteencoding properties in the Formfile component, introduced in Studio 4.0, were changed in Studio 4.1 (and above), whereby the kFFEncoding… constants were deprecated and replaced by the kUniType… constants to identify the file encoding. Formfile now uses any of the kUniType... constants except kUniTypeCharacter for the $filereadencoding property, and any of the kUniType... constants except kUniTypeAuto and kUniTypeCharacter for the $filewriteencoding property.

In addition, there is also a kUniTypeBinary constant to identify files that are to be treated as raw binary data. Code that uses the old constants should continue to work.

Fileops

The Fileops component has two new methods, $readcharacter() and $writecharacter(), which allow you to read and write Unicode character data from and to a file.

- $readcharacter(encoding,variable)
  reads all data from a file containing character data into variable; encoding is one of the kUniType… constants (listed above), identifying the encoding of the file.
- $writecharacter(encoding,variable)
  replaces the contents of the file with the character data stored in variable; encoding is one of the kUniType… constants, identifying the encoding of the file.

For $readcharacter, specify the encoding as any kUniType... constant except kUniTypeBinary and kUniTypeCharacter. The kUniTypeAnsi... constants are only supported in Unicode Studio.

For $writecharacter, specify the encoding as any kUniType... constant except kUniTypeAuto, kUniTypeBinary and kUniTypeCharacter. The kUniTypeAnsi... constants are only supported in Unicode Studio.

Import/Export and Report File Encoding

You can control the encoding of import text files, export files, and report data written to text files and the port using the $importencoding and $exportencoding properties:

- $importencoding
  The encoding used for imported data when importing from port, or when the import file does not have a Unicode BOM. Any of the kUniType... constants, except kUniTypeAuto, kUniTypeCharacter, kUniTypeBinary and the kUniTypeUTF32… values.
- $exportencoding
  The encoding used for exporting data and printing to port or text file. Any of the kUniType... constants, except kUniTypeAuto, kUniTypeCharacter and kUniTypeBinary.
- $exportbom
  If true, and the $exportencoding preference identifies a Unicode encoding, a Unicode BOM is output at the start of the output file.

These properties can be found in the Omnis preferences ($root.$prefs). In a multi-threaded server, there is a separate value of each of these properties for each thread.

Library Conversion

When you open an Omnis library in the Unicode version, you are prompted to convert the file. This is a full conversion, including conversion of any characters >= 128. Once a library has been converted to Unicode it will no longer open in the non-Unicode version of Omnis.

Version Control

The Unicode version of the Omnis VCS can build libraries that run only in the Unicode version of Omnis Studio. The non-Unicode version of the VCS can build libraries that will run under the Unicode and non-Unicode versions of Omnis Studio.

Edit field scrollbars

The $righttoleft property was added in a previous version to allow text entry in edit fields to scroll from the right to the left, to support data entry on Arabic machines, for example. This property has been enhanced so that in a multi-line field the vertical scrollbar is displayed on the left of the field.

DAMs

Oracle8 DAM (Unicode only)

There is a new Session property for the Oracle8 DAM (DAMORA8), available in the Unicode DAM only.

- $nationaltonclob
  $nationaltonclob is used to alter the default mapping of Omnis Character and National types. By default, Omnis Character and National fields with a subtype greater than $maxvarchar2 are mapped to the NCLOB datatype. By setting $nationaltonclob to kTrue only National fields with a subtype >$maxvarchar2 are mapped as NCLOBs. Character fields with subtype >maxvarchar2 are mapped as non-Unicode CLOBs. Character fields mapped in this way are subject to data loss/truncation where such fields contain Unicode characters.

Omnis Data File Conversion

WARNING: You should make a secure backup of your Omnis data files before converting them in the Unicode version of Omnis Studio.

The Unicode version of Studio 4.3 (and 4.3.x) has a full data file converter which converts the data in your Omnis data file and rebuilds the indexes. In the Unicode version of Studio 4.1 and 4.2, the data was not converted, simply the indexes were dropped and rebuilt, and the data file was marked as Unicode.

When this full data file conversion takes place, in Studio 4.3 or higher, all data marked as Character is converted. In the case where character data is stored in a binary, for example, text stored in a document file, conversion of this data *does not* take place.

When you access a data file in the Unicode version of Omnis Studio you are asked to confirm that you want to convert the data. After you select Yes, Studio displays a dialog which offers two types of conversion:

- Quick
  whereby the indexes are dropped and rebuilt, but the data is not converted (that is, the same conversion process as Studio 4.2). This is OK for files containing only 7 bit data: Omnis does not check that the file only contains 7 bit data, so it's your responsibility to know whether or not it is safe to run this conversion process.
- Full
  the new conversion process whereby a full conversion of the Character based data takes place.

Data File commands

The *Open data file* and *Prompt for data file* commands have an existing option called "Convert without user prompts". If this is checked, the new conversion dialog is not displayed. There is a new option for these commands, "Quick Unicode conversion" or "Full Unicode conversion", which allows you to select the level of conversion to take place.

Data file conversion

The Unicode version of Omnis Studio 4.3 performs a full conversion of Omnis data files to Unicode, as described above. Bearing in mind that the next major release of Omnis Studio will be for Unicode only, we suggest that you convert your Omnis data files to Unicode using the Unicode version of Omnis Studio 4.3 (or 4.3.x), and report any problems to us as soon as possible, so that any issues can be resolved before the Unicode-only version of Omnis Studio is released.

WARNING: Again, we strongly urge you to make a secure backup of your Omnis data files before opening and/or converting them in the Unicode version of Studio 4.3. You should make a copy of your data and test the Unicode conversion in Studio 4.3 and report any problems to us.

For developers using Unicode Studio 4.3, please check the results of full conversion carefully, and do not discard your backup of the non-Unicode data file until you are satisfied that the data has converted successfully. You could perform some regression tests on your application and data – you should normally do this with a new version of Studio, but when converting to Unicode Omnis, and converting your data files, you need to be especially sensitive to possible data file and indexing issues.

Handling Char & Binary data under Unicode

You cannot concatenate a Character variable to a Binary in the Unicode version of Omnis Studio. The correct method is to use $readfile to read the file into a Binary variable, and then parse the binary variable. Assigning Character to Binary and vice-versa is likely to cause problems, especially in the Unicode version, and should be avoided.

Localisation

Localisation is the process of readying your application for deployment to customers that require your application in a foreign language. In practice, this means translating any text that is visible to the customer into the foreign languages you wish to support, but you may also need to change how data is handled, sorted or displayed for particular regions.

To localise your application, you'll need to change certain things in the Omnis program itself (the runtime Omnis.exe), and all the text strings and labels in your Omnis library. If you are deploying your application via the web, then you only need to translate the text in your library, or more precisely anything that the user is likely to see in their browser.

Omnis Studio has various tools to translate the Omnis program and library files which are described in detail in their respective manuals but are summarised here.

Omnis Program Localisation

Localisation of the Omnis runtime program lets you change the following items, some of which are visible to the end-user, while others effect how character and number data is managed.

- The names of the days of the week, and months of the year.
- Separator characters.
- The text for Yes/No, OK/Cancel, True/False, Am/Pm and On/Off.
- The national sort ordering.

All Omnis libraries share the same set of data, stored in an Omnis data file, called omnisloc.df1 or the Localisation database, which is located in the Local folder under the main Omnis folder. An Omnis library, omnisloc.lbr, lets you create and edit language information in the Localisation database.

The Localisation database (omnisloc.df1) contains a data slot for configuration data; each record in that slot contains a complete set of data corresponding to a language. It also contains a data slot with a single record, which identifies the current language, that is, the current set of configuration data.

The Localisation library (omnisloc.lbs) has been updated for the Unicode version of Omnis Studio. The sort order field is now labelled Locale when running the Unicode version. Also, in the Unicode version only, there is a new check box below the Locale field, "Use Locale For Defaulted Items". This determines the locale used for language items that have been left empty, so that they get a default value from the system:

- If the Use Locale check box is checked, default values come from the locale stored in the language record.
- If it is not checked, default values come from the operating system default locale.

Omnis Library Localisation

You have two options available to translate the text and other labels in your Omnis application:

- **The Translation Library**
  this Omnis library (trans.lbs), located in the Local folder under the main Omnis folder, allows you to translate the text and labels throughout your library; you export all the literals in your Omnis library, translate them, and import them back into your library.
- **String Tables**
  using tables of translated strings, you can dynamically change the text and labels in your Omnis application; you can use the String Table Editor in Omnis to create tables containing a matrix of words for any number of languages and use the string table functions to load translated text as required; the String table editor is located under the Tools>>Add Ons menu

We'd like to hear from you

We'd like to hear from you about any questions you may have about the switch to Unicode-only Omnis. Hopefully we can answer your queries and try to address them in the next major release of Omnis Studio.

And if you've used the Unicode version of Omnis Studio to create an application that supports languages other than English, then we'd like to hear from you as well. We may be able to feature you in our Success Stories (case studies) section on the Omnis website. Please contact us.