



TEMENOS™

## **jBASE Internationalization**



## Copyright

Copyright (c) 2005 TEMENOS HOLDINGS NV

All rights reserved.

This document contains proprietary information that is protected by copyright. No part of this document may be reproduced, transmitted, or made available directly or indirectly to a third party without the express written agreement of TEMENOS UK Limited. Receipt of this material directly from TEMENOS UK Limited constitutes its express permission to copy. Permission to use or copy this document expressly excludes modifying it for any purpose, or using it to create a derivative therefrom.

## Acknowledgements

Information regarding Unicode has been provided in part courtesy of the Unicode Consortium. The Unicode Consortium is a non-profit organization founded to develop, extend and promote use of the Unicode Standard, which specifies the representation of text in modern software products and standards. The membership of the consortium represents a broad spectrum of corporations and organizations in the computer and information processing industry. The consortium is supported financially solely through membership dues. Membership in the Unicode Consortium is open to organizations and individuals anywhere in the world who support the Unicode Standard and wish to assist in its extension and implementation.

Portions of the information included herein regarding IBM's ICU has been reprinted by permission from International Business Machines Corporation copyright 2001

jBASE, jBASIC, jED, jSHELL, jLP, jEDI, jCL, jQL, j1, j2 j3 j4 and jPLUS files are trademarks of TEMENOS Holdings NV.

REALITY is a trademark of Northgate Solutions Limited.

PICK is a trademark of Raining Data Inc.

All other trademarks are acknowledged.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

### **Errata and Comments**

If you have any comments regarding this manual or wish to report any errors in the documentation, please document them and send them to the address below:

Technical Publications Department

TEMENOS UK Limited

2 PeopleBuilding Estate

Hemel Hempstead

Hertfordshire

HP2 4NW

England

Tel : +44 (0) 1442 431000

Fax: +44 (0) 1442 431001

Please include your name, company, address, and telephone and fax numbers, and email address if applicable. [documentation@temenos.com](mailto:documentation@temenos.com)

## Contents

Documentation Conventions .....	i
<b>INTRODUCTION .....</b>	<b>III</b>
WHAT IS INTERNATIONALIZATION?.....	IV
The Development Challenges .....	v
The Development Approaches .....	v
Internationalization and Localization .....	vi
The Process of Internationalizing.....	vi
The Process of Localizing .....	vii
Code Pages .....	vii
Locales.....	vii
Unicode .....	viii
Unicode Combining/Composite Characters.....	ix
Unicode Expanding and Contracting Characters .....	ix
Unicode Normalization .....	ix
Unicode Universal Character Set and UTF-8.....	x
Unicode UTF-8 Properties .....	x
International Components for Unicode (ICU).....	xi
jBASE Internationalization Configuration.....	xii
jBASE Code Page Configuration .....	xiii
jBASE Locale Configuration .....	xiii
JBASE TIMEZONE CONFIGURATION.....	XIV
jBASE Function Changes for International Mode .....	xiv
Character properties .....	xv
Collation properties .....	xvi
Conversion properties.....	xvii
Character functions.....	xviii
Additional Functions .....	xviii
Timestamp Functions .....	xix
Byte Count Functions.....	xix
jBASE JQL Changes for International Mode.....	xx
jQL Dictionary Conversions and Correlatives .....	xx
jQL Locale-Based Collation.....	xxi
jQL Right Justified Sort .....	xxi
Data Conversion .....	xxii
jBASE File Conversion.....	xxii
Conversion Map .....	xxiv
Data Import and Export.....	xxiv
jBASE Error Message Files .....	xxv
jBASE Spooling and Printing .....	xxvi
jBASE OBJEX, JavaOBJEX and JDP .....	xxvi
Potential Performance Issues.....	xxvii
Desktop Applications .....	xxviii
The Future and UTF-8.....	xxviii

Summary .....	xxix
jBASE Environment Variables for Internationalization.....	xxix
Unicode and ICU Reference Information.....	xxx
Locale Definition .....	xxx
Configured Locales.....	xxxi
Language Codes (ISO-639) .....	xxxiii
Country Codes (ISO 3166) .....	xxxvi
Configured Code Pages and Aliases.....	xlvii
UTF and ASCII Code Pages.....	xlvii
Latin Code Pages .....	xlviii
Mircosoft Windows Code Pages .....	xlix
Japanese, Chinese, Korean Code Pages.....	xlix



## Documentation Conventions

This manual uses the following conventions:

Convention	Usage
<b>BOLD</b>	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates JBASE commands, keywords, and options; BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates JBASE identifiers such as filenames, account names, schema names, and Windows NT filenames and pathnames.
UPPERCASE <i>Italic</i>	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, filenames, and pathnames.
Courier	Courier indicates examples of source code and system output.
Courier Bold	<b>Courier Bold</b> In examples, courier bold indicates characters that the user types or keys (for example, <Return>).
[]	Brackets enclose optional items. Do not type the brackets unless indicated.
{ }	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
ItemA   itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
...	Three periods indicate that more of the same type of item can optionally follow.
⇒	A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose <b>File</b> ⇒ <b>Exit</b> ” means you should choose <b>File</b> from the menu bar, and then choose <b>Exit</b> from the File pull-down menu.

Syntax definitions and examples are indented for ease in reading.

All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.

Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

# INTRODUCTION

The release of jBASE 4.1 incorporates modifications to the underlying jBASE library functions and components to provide the tools and mechanisms whereby global communities can internationalize and localize applications. The internationalization functionality provides applications with the capability for correct handling of locale dependent collation sequencing, along with processing of unique character properties, code page data import/export and terminal/printer data input and output. The jBASE library functions when used in International Mode process internally using character rather than byte orientated values and properties such that applications can be easily coded or converted by minimum change for the international market.

## What is Internationalization?

More applications are crossing international and cultural boundaries, which require localization to provide support for the local language of the user. Internationalization is the development of software for localized user communities without the need to change the executable code.

Fundamentally, computers deal with numbers. They store letters and other characters by assigning a number for each one. Before the invention Unicode, there were hundreds of different encoding systems for assigning these numbers. No single encoding could contain enough characters: for example, the European Union alone requires several different encodings to cover all languages. Even for a single language like English, no single encoding was adequate for all the letters, punctuation, and technical symbols in common use.

These encoding systems also conflict with one another. That is, two encodings can use the same number for two different characters, or use different numbers for the same character. Any given computer may need to support many different encodings; yet, whenever passing data between different encodings or platforms, that data always runs the risk of character corruption from incorrect or incompatible conversion.

Internationalizing allows applications to:

- Provide applications for global markets
- Interface to internationalized processes
- Increase portability
- Reduce development costs

What needs to be internationalized?

- Terminal and File I/O
- Dates / Times
- Numbers / Currency
- Searching / Sorting
- Messages / Properties

How to go about internationalizing an application:

- Separate Executable Code from User Interface
- Handle Numbers, Currency, Dates and Times independently of formatting

- Provide for Text Input and Layout
- Allow for Alphabetical Order of Characters
- Allow for Character Format
- Avoid Cultural Assumptions

## **The Development Challenges**

As different countries use different characters to represent words, some basic problems arise in the context of software development when considering the global market. Here follows some common problems developers need to consider:

- The basic concern is that there are more than 256 characters in the world. Languages such as Cyrillic, Hebrew, Arabic, Chinese, Japanese, Korean, and Thai use characters that are not included in the 256-character ASCII set; but somehow, these characters must be made available.
- It is impossible to store text from different character sets in the same document/record: if each document has its own character set, manual intervention for conversion becomes inevitable.
- The introduction of new characters, such as Euro symbols must be accounted for. The Euro is replacing old European currency symbols; documents containing these symbols have now changed.

How can applications interchange data that may include one or more character sets? The solution is to adopt a worldwide usable character set and use a well-conceived strategy that incorporates that character set to produce software products.

## **The Development Approaches**

Companies often develop a first version of a program or system to just deal with English. Later, when it becomes necessary to produce the first international version, the product is 'internationalized' by going through all the lines of code, translating the literal strings. The process is time consuming and is not a good long-term strategy. Each new version is expensive, as the painstaking process to identify all the strings, which require translation is repetitive. Because there are multiple versions of the source code, maintenance and support becomes very expensive. Moreover, there is a high probability that a translator may introduce bugs by mistakenly modifying code.

The fundamental advantage of internationalizing software is that the developer is not limited to the number of different languages or countries that you can support. If, after developing the internationalized product and you need to provide the software in another language, it is simply a matter of translating resource files into that language.

By truly internationalizing the product, the software developer is able to:

- Provide applications to global markets;
- Interoperate with external processes;
- Increase systems portability; and
- Reduce development costs.

For applications that will only use the standard ASCII characters, internationalization is not a vital concern. However, for those applications that need to handle characters outside the ASCII range, internationalization is the best medium- and long-range development solution.

## **Internationalization and Localization**

*Internationalization is the process of producing a globalized product, in terms of both the design and the code, which is independent of the language, script, and culture.*

The design of Internationalized products is to support any language, script, culture, and code pages, through the localization process, with minimal expense and effort. The localization of an internationalized product is possible without any code change, merely by translating text and perhaps supplying new graphics.

When internationalizing a software product, the goal is to develop software independently of the countries or languages of its intended users. The translated software is ready for multiple countries or regions. This completes and reduces the Localization time as the process of internationalization through eliminating the need to examine the source code in order to translate the displayed user strings; it translates the resource files containing the display strings.

## **The Process of Internationalizing**

The overriding goal of internationalizing programs is to prepare and ensure the code never needs modification; separate files contain the translatable information. This process involves a number of modifications to the code:

- Move all translatable strings into separate files called resource files, and make the code access those strings when needed. These resource files are completely separate from the main code, and contain nothing but the translatable data.

- Change variable formatting to be language-independent.
- Change sorting, searching, and other types of processing to be language-independent. This means that dates, times, numbers, currencies, and messages call functions to format according to local language and country requirements.

Concluding the process gives you an internationalized product.

## **The Process of Localizing**

Localization is the process of adapting an internationalized offering to a specific language, script, and culture. A localized product is one that is fully adapted to a country's language and cultural conventions, such as currency, number formats, sorting sequence, and so on.

Localizing an internationalized program involves no changes to the source. Instead, contractors and/or translation agencies modify the files.

The initial cost of producing internationalized code is somewhat higher than localizing to a single market, but the advantage is that you only pay that cost once. The cost of doing an additional localization, once the code is internationalized, is a fraction of the previous cost, and avoids the considerable cost of maintenance and source code control for multiple code versions.

## **Code Pages**

The term 'code page' refers to any of the many different schemas and standards used to represent character sets for various languages. Unicode has combined the various code pages into the Unicode Standard (which is equivalent to ISO 10646).

Examples of languages and their corresponding code pages:

English:	ASCII
French, German:	Latin1, Windows1252
Cyrillic:	Latin5, Windows1251
Chinese, Japanese, Korean, and Vietnamese:	CJKV, Win950, Win932, Win949, Win1258

## **Locales**

From a geographical perspective, a locale is a place. From a software perspective, a locale is a set of information associated with a place. The locale information includes the name and identifier of

the spoken language, sorting and collating requirements, currency usage, numeric display preferences, and text directionality (left-to-right or right-to-left, horizontal or vertical).

Up to three different parts reference specified locales:

- Language code
- Country Code
- Variant Code

Depending on the locale, you can specify

- Only the language code,
- The language code and country code
- Or the language code, country code and variant.

Each locale specifies a resource bundle of localized text, for example:

`fr_FR_EURO`

The 'fr' is the French language code, the 'FR' is the country code; the EURO signifies the use of euro currency.

## Unicode

Unicode is a single-coded character set providing a repertoire for all the languages of the world. Its first version used 16-bit numbers, which allowed encoding for 65,536 characters; further development allowed a repertoire of more than one million characters, requiring 21 bits. Higher bits have been declared unusable to ensure interoperability between UTF encoding schemes; UTF16 cannot encode any code points above this limit. The handling of values above 16 bits is by two 16-bit codes.

- The first Unicode version used 16 bits, which allowed for encoding 65,536 characters.
- Further extended to 32 bits, although restricted to 21 bits to ensure interoperability between UTF encoding schemes.
- Unicode provides a repertoire of more than one million characters.

The 16-bit subset of UCS (Universe Character Set) is known as the Basic Multilingual Plan (BMP) or Plane 0.

Unicode provides a unique number for every character, on every platform, for every program, no matter what the language. Standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., requires Unicode and is the official way to implement ISO/IEC 10646; supported in many operating systems, all modern browsers, and many other products. Incorporating Unicode into client-server or multi-tiered applications and websites can offer 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, and allows data to be transported through many different systems without corruption.

## Unicode Combining/Composite Characters

Some code points have been assigned to combining characters. A combining character is not a full character by itself; rather, it is an accent or other diacritical mark added to the previous character. This makes it possible to add an accent or other on any character. This capability is particularly important for scientific notation in mathematics.

## Unicode Expanding and Contracting Characters

Some languages treat certain single characters as two separate characters for collating purposes (expanding), whereas some languages also treat two characters as a single character when collating (contracting).

**Contracting** In Spanish sort order, ‘ch’ is considered a single letter. All words that begin with ‘ch’ sort after all other words beginning with ‘c’

**Expanding** In German, ä is equivalent to ‘ae,’ such that words beginning with ä sort between words starting with ‘ad’ and ‘af’.

## Unicode Normalization

**Normalization** is the removal of ambiguities caused by precomposed and compatibility characters. There are four different forms of normalization.

Form D..... Split up (decompose) precomposed characters into combining sequences where possible.

Form C..... Use precomposed instead of combining sequences where possible.

Form NFKD ..... Like D but avoid use of compatibility characters (e.g., use ‘fi’ instead of U+FB01 LATIN SMALL LIGATURE FI).

Form NFKC..... Like C but avoid use of compatibility characters.

Below is an example of Precomposed vs. Decomposed characters.

Precomposed	ü = U+00FC
Decomposed	ü = U+0075 + U+0308

**Note:** that UTF-8 encoding requires the use of precomposed characters wherever possible.

## Unicode Universal Character Set and UTF-8

The Universal Character Set, UCS assigns each character not only a code number, but also an official name. A U+ precedes a hexadecimal number representing a Unicode Code Point or (e.g., U+0041 'Latin Capital Letter A').

UTF-8 can represent all possible Unicode code points by byte sequences, which in turn represent different code points. The sequence used for any given character depends on the Unicode number, which represents that particular character. The Universal Character Set has the following properties:

- UCS U+0000 to U+007F is identical to US-ASCII (IISO 646 IRV).
- UCS U+0080 to U+00FF are identical to ISO-8859-1 (Latin-1)
- UCS U+E000 to U+F8FF is reserved for private use.

UTF-8 encoding is a Unicode Translation Format of Unicode. Before UTF-8 emerged, users all over the world had to use various language-specific extensions of ASCII. This made the exchange of files difficult, and application software had to consider small differences between these encodings. Support for these encodings was usually incomplete and unsatisfactory, because the application developers rarely used all these encodings themselves.

## Unicode UTF-8 Properties

UTF-8 encoding avoids several problems inherent in other encoding systems:

- Files and strings that contain only 7-bit ASCII characters have identical encoding under ASCII and UTF-8.
- ASCII bytes 0x00-0x7F cannot appear as part of any other character.
- Allows easy resynchronization and makes the encoding stateless and guards against the possibility of missing bytes.

- Can encode all possible 231 UCS codes.
- UTF-8 encoded characters may theoretically be up to six bytes long, however 16-bit BMP characters are only up to three bytes long.
- The sorting order of Bigendian UCS-4 byte strings is preserved.
- It never uses the byte 0xFE and 0xFF in the UTF-8 encoding.
- UTF-8 is also much more compact than other encoding options, because characters in the range 0x00-0x7f still only use one byte.
- Use only the shortest possible multi byte sequence that can represent the code point of the character.
- In multi byte sequences, the number of leading one bit in the first byte is identical to the number of bytes in the entire sequence.
- Unicode represents a unique character by a unique 32-bit integer. Hence using UTF-8 encoding avoids problems, which would arise if using 16 or 32 bit character byte streams, as the normal C string termination byte is a zero, thus byte streams could become prematurely truncated.

## **International Components for Unicode (ICU)**

International Components for Unicode (ICU) is IBM's open source package for cross-platform Unicode library enablement for C/C++ products. ICU provides functions for formatting numbers, dates, times, currencies according to locale conventions, and similarly, ICU provides code and data to handle the complexities of native language collation, searching, and other processes. It also provides a mechanism for accessing strings from resource files, whereby sharing common strings across countries that have the same language.

Perhaps the chief benefit of ICU is that it provides fully portable cross-platform libraries. Since the code is portable to a wide variety of platforms, it is possible to share data formats that drive the code, at runtime, across different platforms.

JBASE has implemented additional functions that interface with the ICU API's; for example, instead of using normal standard jBASE date conversions, it invokes the ICU date conversion procedures, thereby providing fully internationalized date formats.

## **jBASE, Internationalization and UTF-8**

The code of jBASE 4.1 releases provides internationalization functionality, sometimes referred to 'i18n', (i followed by the 18 characters of nternationalizatio followed by n). This enables applications to take advantage of the internationalization functionality and hence provide for the global market, i.e. a fully internationalized application.

When configuring application accounts for international mode, all program variables and data records are to be of type UTF-8 handled internally as UTF-8 encoded byte sequences. The UTF-8 encoding scheme manipulates characters other than those in the standard 7-bit ASCII range (0x00-0x7f) as two-byte or three-byte sequences, rather than the normal single 8-bit byte.

The number of bytes in the UTF-8 sequence will depend on the original code page. For example, it encodes characters in the range 0x80-0xff, representing the single byte character set ISO-8859-1, as two bytes. However, it represents characters imported from a Double Byte Character Sequences (DBCS), such as Kanji characters from the Japanese code page “shift\_jis”, as three bytes when encoded as a UTF-8 byte sequence.

When executing in international mode, all terminal input can be configured to be converted from the configured input code page to a UTF-8 byte sequence. Similarly, for terminal output, configure the UTF-8 byte sequences such that it converts the output to one of the output code pages dependent upon the code page of the terminal device. Normally the input and output code pages would be the same. There is also an obvious advantage to skipping the conversion step and using UTF-8 direct where possible to communicate with terminal devices as these helps reduce the conversion overhead from UTF-8 to the configured code page. Several telnet emulators now support a UTF-8 mode for telnet communication.

UTF-8 is an ASCII preserving encoding of the ISO/IEC Unicode Standard 10646. Therefore, all ASCII characters remain the same and still use a single byte of storage. This provides UTF-8 encoding with an advantage over other forms of Unicode encoding or translation formats. Some forms would require either a doubling (UTF-16, UCS2) or quadrupling (UTF-32, UCS4) of byte space required to represent the Unicode Code Point; however with UTF-8 encoding, only the characters over and above the ASCII 7 bit range need to be stored as multi bytes.

## **jBASE Internationalization Configuration**

JBASE 4.1 provides internationalization support code page conversion, collation sequences, international dates and times along with number and currency formatting. The basis of the internationalization configuration is on user id and/or certain jBASE environment variables:

- a. JBASE\_CODEPAGE,
- b. JBASE\_LOCALE,
- c. JBASE\_TIMEZONE.

NOTE: The user id configuration or environment variables have no effect if the account in which it executes the application is not configured for international mode or the environment variable JBASE\_I18N is not set.

Application providers are responsible for the handling of all directionality issues, such as left-to-right, top-to-bottom orientation. The jBASE library functions such as length (LEN); string

comparisons (LT, LE, GT, GE) and collation order statements like (LOCATE/SORT) have all been modified to operate on a character basis in international mode rather than bytes, along with the currently configured user locale.

## **jBASE Code Page Configuration**

*Configure the Code pages for the user id using the `JBASE_CODEPAGE` environment variable.*

*Display a full list of available code pages using the “`jcodepages`” command.*

All input and output conversion can be undertaken, however it is more efficient to use UTF-8 for input and output if possible, as no code page conversion is then necessary, reducing system resource requirements. There are several commercially available telnet clients that can communicate using UTF-8, in these cases the telnet client performs the conversion from the configured code page to UTF-8, hence it is important to ensure that the client is configured correctly such that the input and output code page is the correct one for the keyboard mapping required.

Code page conversion is only applicable when the `JBASE_I18N` environment variable is set. If the `JBASE_I18N` environment is not set, then code page conversion will *not* occur, and all variables will be handled as bytes rather than as characters. As configuration of international mode is on an account basis the state of international mode can change on execution of a LOGTO.

## **jBASE Locale Configuration**

Locales can be configured for the user id via the `JBASE_LOCALE` environment variable.

Display a full list of available locales from the command line by the “`jlocales`” command.

Configured locales are only applicable when executing an application in international mode or the `JBASE_I18N` environment variable is configured. The locale is based on the underlying OS locale configuration and the configured locale for the user id has no effect.

As configuration of the international mode is on an account basis, the state of international mode can change on execution of a LOGTO. If configuring an account with international mode ‘false’ then the `JBASE_I18N` environment variable will be unset as the result of the LOGTO.

## **jBASE Timezone Configuration**

Timezones can be configured for the user id via the `JBASE_TIMEZONE` environment variable. Display a full list of available Timezones from the command line by the “`jttimezones`” command. Configured Timezones are only applicable when executing an application in international mode or when the `JBASE_I18N` environment variable is configured. If the `JBASE_I18N`, environment variable is not set, the timezone is based on the underlying OS timezone configuration and the configured timezone for the user id has no effect. As configuration, the international mode on an account basis the state of international mode can change on execution of a LOGTO. If configuring an account with international mode ‘false’ then the `JBASE_I18N` environment variable will be unset as the result of the LOGTO.

## **jBASE Function Changes for International Mode**

Internally, very few of the jBASE library functions need changing to process data as UTF-8 encoded multi byte sequences rather than single bytes. It bases resultant values on characters rather than bytes. Some functions change their internal workings depending upon the state of international mode or `JBASE_I18N` setting.

### **Character vs. Bytes**

#### **LEN, SUBSTRINGS, X[n,m], INDEX**

In international mode, length and sub string extraction works in ‘characters’ not ‘bytes’. Resultant positions are character positions not byte offset.

#### **BYTELEN**

A new function has been provided to obtain the actual number of bytes rather than characters. For example:

The following source code example contains UTF-8 encoded characters representing the German ‘u’ umlaut (0xC3 0xBC) and the double ‘s’ (0xC3 0x9F).

```
X = "Fußball";* String as UTF-8 sequence "F.C3.BC.C3.9Fball"
CRT X
CRT "Character Length of X is ":LEN(X)
CRT "Byte Length of X is ":BYTELEN(X)
CRT "Substring[1,3] of X is ": X[1,3]
```

If executed in international mode and with the Input/Output Code Page configured to ISO-8859-1 (Latin1) this code will produce the following output.

**NOTE:** the length returned by the LEN function is the number of characters in variable X, whereas the length returned by the BYTELEN function is always the number of bytes in variable X.

Fußball

Character Length of X is 7

Byte Length of X is 9

Substring[1,3] of X is Füß

## Character properties

### UPCASE, DOWNCASE, ALPHA, MATCHES, MATCHFIELD

In international mode, functions use the configured locale to convert and/or test character properties. For example:

The following source code example contains a UTF-8 encoded byte sequence representing the German 'u' umlaut (0xC3 0xBC).

```
X = "ü"      ;* this string held in source as UTF-8 "C3.BC"
CRT X: " becomes ": UPCASE(X)
IF ALPHA(X) THEN CRT X: " is alphabetic "
IF X MATCHES "1A" THEN CRT X: " is alphabetic "
```

If executing in international mode and with the Input/Output Code Page configured to ISO-8859-1 and with the locale configured for German (de\_DE) the code produces the following output.

ü becomes Ü

ü is alphabetic

ü is alphabetic

The UPCASE function converts the lower case u umlaut to the upper case equivalent. In other words, the UTF-8 byte sequence 0xC3 0xBC becomes 0xC3 0x9C.

The ALPHA function tests the lower case u umlaut as an alphabetic character according to the configured locale, de\_DE.

The MATCHES statement tests the lower case u umlaut against the single alphabetic character according to the configured locale, de\_DE.

## Collation properties

### **SORT, LOCATE, COMPARE, LE, LT, GE, GT**

In international mode, statements use the configured locale to determine sort order. For example: A sort of the following UTF-8 encoded byte sequences using the SORT function will generate a different sort order depending on the configured locale.

locale configured for 'en\_US'

cote	stored as UTF-8 sequence 'cote'
coté	stored as UTF-8 sequence 'cot.C3.A9'
côte	stored as UTF-8 sequence 'c.C3.B4te'
côté	stored as UTF-8 sequence 'c.C3.B4t.C3.A9'

locale configured for 'fr\_FR' (reverse accented collation)

cote  
côte  
coté  
côté

**Note:** that the word côte sorts BEFORE the word coté for the configured locale fr\_FR

```
X = "côte"      ;* Source contains UTF-8 sequence "c.C3.B4te"  
Y = "coté"     ;* Source contains UTF-8 sequence "cot.C3.A9"
```

The statement :

```
IF X LT Y THEN CRT X:" is lower in collation sequence than ":Y
```

It produces the following output in International mode when executed with the locale configured for French, fr\_FR.

côte is lower in collation than coté

## Conversion properties

### ICONV, OCONV, FMT

The implementation of conversions is by a set of jBASE library functions, which in turn invoke functions in the IBM Public License package, ICU. This package provides cross-platform open source libraries compliant with Unicode Standard 3.0 and currently supports over 170 locales independently of the system locales. Several input and output conversions become dependent upon the configured locale.

For example, then following source code example will output a different date format dependent upon the configured locale when executing in international mode.

```
CRT OCONV(0, "D2/ ")  
CRT OCONV(0, "D ")
```

This code will produce the following if executed in international mode with a configured German locale of 'de\_DE'.

```
31/12/67  
31 DEZ 1967
```

However, some conversions can be used to 'force' an expected format regardless of locale, for instance the DE date format will always produce a European date format. The DG format is a new Global date format for YYYYMMDD.

```
CRT OCONV(0, "D2/E")           displays           31/12/67  
CRT OCONV(0, "DG")            displays           19671231
```

## Character functions

### **CHAR, SEQ**

The CHAR and SEQ functions behave differently for international mode.

In international mode the CHAR function now provides for an extended numeric range to support 32 bit Unicode code point values. The CHAR function will return a UTF-8 encoded byte sequence for the numeric range 128-247 (0x80-0xf7) and the range 256 and beyond, however numeric values in the system delimiter range 248-255 (0xf8-0xff) will continue to return the normal single byte system delimiters characters. The resultant characters for numeric values in the ASCII range 0-127 (0x00-0x7f) are unchanged.

In international mode, the SEQ function now accepts UTF-8 encoded byte sequences such that UTF-8 byte sequences representing characters in the range 0-127 (0x00-0x7f), i.e. single byte characters return the normal ASCII numeric values. UTF-8 encoded byte sequences representing characters in the range 128-255 (0x80-0xff) will return the ISO-8859-1 equivalent numeric value. System delimiter characters will return numeric values in the range 248-255 (0xf8-0xff). Other UTF-8 encoded byte sequences will return the equivalent numeric value as specified by the Unicode code point.

## Additional Functions

### **BYTELEN, LATIN1, LENDP, UTF8**

The provision of additional functions helps with programs that need to know the actual real byte length of a variable as well as conversion functions for handling binary values. The conversion function should only be required when dealing with binary data, for example handling data to/from tape devices.

The BYTELEN function returns the number of actual bytes used for the string variable. Use this function whether executing in international mode or not.

The LATIN1 function will convert a string variable from ISO-8859-1 to a UTF-8 encoded byte sequence. Use this function whether executing in international mode or not.

The LENDP function will return the number of character display positions required in order to display the string variable. Use this function to determine the display width of characters, for instance the null character has a display width of zero, whereas some Japanese Kanji characters

require more than one display position. The LENDP function will change behaviour if used without international mode set true.

The UTF8 function will convert a string variable from UTF-8 encoded byte sequence to the ISO-8859-1 (binary) equivalent. Use this function whether executing in international mode or not.

## Timestamp Functions

### **TIMESTAMP, TIMDIFF, CHANGETIMESTAMP, MAKETIMESTAMP**

### **LOCALDATE, LOCALTIME**

The provision of additional functions assist with date and time internationalisation; these functions enable applications to obtain, convert and process a 'timestamp'. These functions are available regardless of current state of international mode.

The **TIMESTAMP** functions returns a timestamp of Universal Coordinated Time, UTC as decimal seconds.

The **TIMEDIFF** functions returns the interval between two timestamps.

The **CHANGETIMESTAMP** function generates a new timestamp by adjusting the supplied timestamp by a dynamic array, which specifies the adjustment values.

The **MAKETIMESTAMP** function generates a timestamp using a specified time zone.

The **LOCALTIME** function generates an internal time value using a supplied timestamp and time zone.

The **LOCALDATE** function generates an internal date value using a supplied timestamp and time zone.

## Byte Count Functions

### **READBLK, WRITEBLK, OSBREAD, OSBWRITE**

The prime target of the statements **READBLK** and **WRITEBLK** are at device access and hence use a block size or byte count. It is normal for device formats to use binary values to describe the contents of the data blocks regardless of the underlying structure. As such, these statements continue to work on a byte rather than character basis whether used with international mode set true or not.

If the requirement is to read/write large files, use instead the **READSEQ/WRITESEQ** commands.

In the default configuration the **READSEQ** and **WRITESEQ** statements read/write a line at a time such one a line from the file has been read into a variable, that variable can be used on a character basis rather than bytes. This assumes that the data in the file is UTF-8 encoded. If the data in the file is not UTF-8 encoded, but ISO-8859-1 (binary) then convert the data to UTF-8 using the **UTF8** function.

**Note:** if using IOCTL commands to suppress one line at a time mode for the READSEQ or WRITESEQ, operates these statements only in byte mode, and not character mode

## **jBASE JQL Changes for International Mode**

Modification of the jBASE jQL Processor in several areas provides full Internationalization capabilities.

### **jQL Dictionary Conversions and Correlatives**

For dates and times, it applies simple date format functions to use the configured locale to support the standard conversions D and MTS. Formatting numbers via MR/ML/MD, use locale for Thousands, Decimal Point and Currency notation.

TimeStamp "W{Dx}{Tx}"

In addition, it includes a provided suite of conversions including A, F and I-types for timestamp functionality. This is such that it displays a generated timestamp for date and/or time in short, long, and full formats. These conversions also support non-Gregorian locales. The meaning of the components of the conversion is as follows:

- W - Is a new conversion code so not to clash with existing conversions.
- D - Date
- T - Time
- x - Format option: S = Short, M = Medium, L = Long, F = Full

"WDS" or "WTS" SHORT is completely numeric. **12/13/52 or 3:30pm**

"WDM" MEDIUM is longer. **Jan 12, 1952**

"WDL" or "WTL" LONG is longer. **January 12, 1952 or 3:30:32pm**

"WDF" or "WTF" completely specifies FULL.

## **jQL Locale-Based Collation**

As a part of the internationalization of jBASE, jQL will now use collation tables, when enabled for international mode, that are specific for the user's locale.

When international mode is not enabled, the keys are sorted by the binary value of the individual characters as in prior releases.

When international mode is enabled, the keys are first passed to a lookup algorithm that converts the key into a collation key, which is tailored specifically for the user's language. Using the collation key, the sort processor is able to produce output in the order expected in the user's locale.

## **jQL Right Justified Sort**

The primary purpose of right justified attribute definitions is to produce the correct sort sequence and display properties for numeric values; use also for mixed alpha and numeric fields to produce a meaningful sort order.

The use of right justified fields with completely non-numeric data does not affect sort order, just the display.

As part of the internationalization of jBASE, jQL uses a new algorithm for right justified fields designed to provide optimal sorting of mixed alpha and numeric fields as well as numeric fields. The field width specified in the attribute definition no longer affects the behaviour of the sort.

## **Pure Numeric keys**

It sorts Keys from the largest negative number to the largest positive number.

A single leading minus or plus sign may be present, which ignores leading zeros before a decimal point for sorting purposes. It ignores trailing zeros after a decimal point for sorting purposes.

Nulls will sort either before all numeric keys or as zero, depending on emulation option. If international mode is true, the defined characters in the Unicode 3.0 specification (section 4.6) to be decimal digits are sorted as numbers.

## **Mixed Alpha Numeric Sorting**

The goal here is to produce a meaningful sort order of a field of mixed alpha and numeric data, for example, a field containing a suppliers' part number.

In this format and content, the field is unknown and can be expected to contain alpha, alphanumeric, and pure numeric values. Each candidate key is split into parts, alternating between numeric and non-numeric parts. Sign characters are only valid as the first character of the key, elsewhere they are treated as non-numeric. If the part is numeric then it processes that part in the same manner as a pure numeric key above. If international mode is true, it passes non-

numeric parts through the collation algorithm to produce collation key parts. If international mode is false, it sorts the non-numeric parts left to right.

## Data Conversion

When executing programs in international mode, it processes all variable contents as UTF-8 encoded sequences. As such all data must be held as UTF-8 encoded byte sequences. This means that data imported into an Account configured to operate in international mode must be converted from the data's current code page to UTF-8. Normally if ALL the data are 8 bit bytes in the range 0x00-0x7f (ASCII) then no conversion is necessary as these values are effectively already UTF-8 encoded. However values outside of the 0x00-0x7f range must be converted into UTF-8 proper such that there can be no ambiguity between character set code page values.

For instance, the character represented by the hex value 0xE0 in the Latin2 code page, (ISO-8859-2), is described as "LATIN SMALL LETTER R WITH ACUTE". However the same hex value in the Latin1 code page, (ISO-8859-1), is used to represent the character "LATIN SMALL LETTER A WITH GRAVE".

To avoid this clash of code pages the Unicode specification provides unique hex value representations for both of these characters within the specifications 32-bit value sequence.

### EXAMPLE

Unicode value 0x00E0 used to represent LATIN SMALL LETTER A WITH GRAVE

Unicode value 0x0155 used to represent LATIN SMALL LETTER R WITH ACUTE

**NOTE:** UTF-8 is an encoding of 32 bit Unicode values, which also has 'special' properties (as described earlier), which Unix and Windows platforms can use effectively.

Another good reason for complete conversion from the original code page to UTF-8 is that doing so also removes the requirement for 'on the fly' conversions when reading/writing to files, as this would add massive and unnecessary overhead to ALL application processing, whereas the conversion from original code page to UTF-8 is a 'one off' cost.

## jBASE File Conversion

The first requirement before configuring an account and application for international mode is to convert the file data from the original code page into UTF-8 encoded byte sequences.

## Compiler

You must convert all source files containing characters in the range 0x80 thru 0x255 such that these characters are represented in UTF-8 before compiling.

## Conversion Utility

A conversion tool 'jutf8' has been provided to help with the file conversion. The first would be to restore the data in the normal way using a restore process working in binary mode. Once the files have been restored, use the following utility with the imported data files to convert the data. The syntax of the conversion utility is as follows:

```
jutf8 {-options} {filename {,...} }
```

Where options can be:

c	The code page to use for conversion, default latin1
d	Process directories
f	Force mode, skip prompt for confirmation
-m	Use specified map file for conversion
MapFilePath	
-s	Skip sample testing for file already converted
-u	Convert from UTF-8 to code page, i.e., reverse conversion
-v	Verbose mode

The conversion utility, by default, will attempt to confirm that the data is not already converted into UTF-8. Directories are skipped by default unless the -d option is explicitly specified.

**NOTE:** the conversion of file contents containing binary data such as compiled programs may render the compiled object no longer usable. It is recommended that the files be cleared of program object files before use of the utility on source files.

## Conversion Map

Use the MapFilePath option to specify a file that describes the mapping of certain characters, e.g., system delimiters, from and to the required hex value.

The map file describes how characters in the original file should be ‘mapped’ from their current hex value to the required hex value BEFORE conversion to UTF-8 proper. The example below maps any characters in the range 0x01-0x08 into what would normally be system delimiters before conversion to UTF-8. Therefore, character 0x04 is mapped to 0xFC and then converted to the two-byte UTF-8 encoded sequence 0xC4 0xBC, which does not clash with the system delimiter and which in turn represents the 32-bit Unicode value of 0x00FC.

MyMapFile	
#From	To
0x01	0xFF
0x02	0xFE
0x03	0xFD
0x04	0xFC
0x05	0xFB
0x06	0xFA
0x07	0xF9
0x08	0xF8

NOTE: If the map file is specified along with the ‘u’ option then it reverses mapping from/to.

## Data Import and Export

The jBASE Directory and SEQ drivers have been modified to provide for an additional IOCTL command, which provides for data conversion from a specified code page to UTF-8 when reading from the native operating system file. This command can also be used when writing to the native file such that the data is converted from UTF-8 to the configured code page. This IOCTL has been specifically developed for Import and/or Export of data to external applications

and is not recommended for usage as part of an application for 'on the fly' conversion. You can also use this IOCTL with the READSEQ and WRITESEQ statements.

Below is an example of using the IOCTL to convert data in a UNIX directory file from 'shift\_jis', Japanese, to UTF-8 while reading the record from the native file. The record is then written out (without conversion) to a jBASE Hash File. This IOCTL command will also return the previously configured Code Page for the File Descriptor. Note: hash files do *not* support this additional IOCTL command.

\* Convert directory record from CodePage shift-jis to UTF-8 and place into Hash file

```
INCLUDE JBC.h
OPEN 'MYDIRECTORY.' TO FILE ELSE STOP
OPEN 'MYHASHFILE' TO HASHFILE ELSE STOP
```

\* Setup Code Page for IOCTL command

```
CodePage = "shift-jis"
IF IOCTL(FILE, JIOCTL_COMMAND_SETCODEPAGE, CodePage) ELSE
    CRT "Code page problem" ; STOP
END

IF CodePage NE "" THEN CRT "Previously configured Code Page :
":CodePage
* Read and convert record from code page shift-jis to UTF-8
READ Record FROM FILE, "MyCodePage" THEN
    CRT "No Chars ":LEN(Record), "No Bytes ":BYTELEN(Record)
    WRITE Record ON HASHFILE, "MyUTF8"
END
```

## **jBASE Error Message Files**

When executing in international mode, error messages files use the configured locale such that it generates de-nationalized error message files used in place of the default error message file.

The detection of the correct error message file for the locale works on the basis that if the error message for the full locale specification, i.e., it cannot open

LanguageCode\_CountryCode\_Variant, and the process defaults to use the

LanguageCode\_ContryCode. If this still fails, it only uses LanguageCode until ultimately it uses no part of the locale to detect the error message file.

For instance, if configuring a user for the locale 'fr\_FR\_EURO' then any error messages for processing are initially searched for in the "jbcmessages\_fr\_FR\_EURO" file.

If this file cannot be opened, the process will attempt to open "jbcmessages\_fr\_FR". Similarly, if this file is not available, the process will then attempt to open "jbcmessages\_fr". If the open still fails, it uses the default error message file "jbcmessages".

## **jBASE Spooling and Printing**

### **Spooling**

The jBASE spooler files will hold the created spooler jobs as UTF-8 encoded byte sequences only if generated by a program executing in international mode. i.e, as per the Account definition. Else, it creates spooler jobs in the normal Latin1 (ISO-8859-1) code page as previously.

### **Printing**

You can configure a new parameter, CODEPAGE, in the FORM TYPE configuration file in the jBASE release sub directory 'config', (see jspform\_deflt), to specify a code page to use for conversion when despooling the print job. The syntax of the parameter is as follows:

#### **CODEPAGE codepage**

Where "codepage" is the name of the code page to use, such that the print job is converted from the internal format of UTF-8 encoded byte sequences to the required code page for the printer device. For example:

```
CODEPAGE shift-jis
```

This code page parameter will convert the UTF-8 byte sequence in the print job to shift-jis for Japanese.

**NOTE:** the internal format MUST always be UTF-8 if using CODEPAGE parameters; otherwise, fatal conversion errors can occur. If the CODEPAGE parameter is not specified, output will be not be converted, hence if the spool job was generated by a process executing in international mode, then output will be in UTF-8, otherwise if the job was generated by a process executing in normal mode, output will be in ISO-8859-1 (latin1).

Whenever possible, printers should be configured to support UTF-8, so that code page conversion is not necessary, thereby further reducing unnecessary conversion overheads on the system.

## **jBASE OBJEX, JavaOBJEX and JDP**

Within OBJEX, JavaOBJEX and jDP it handles strings as Unicode such that conversions are already handled from the code pages to Unicode, and vice versa. Strings are INSERTed and

REPLACEed using the OBJECT methods, so there is no difficulty with regard to conversion of system delimiter characters.

Similarly, jDP returns only column information to ODBC and OLE DB in Unicode and hence does not return system delimiters, so similarly there are no issues related to jDP and system delimiter conversion.

However, as a precaution for handling as yet unforeseen problems involving system delimiters, support has been added to convert system delimiters to private use Unicode code points, when converting strings between UTF-8 and Unicode: e.g. 0xf8 – 0xff become 0xf8f8 – 0xf8ff.

## Potential Performance Issues

By operating in international mode, it is inevitable that certain functions will suffer in terms of application performance:

- LEN: must now scan variables counting characters, not simply return the number of bytes
- LOCATE: must use the locale for the sort order
- SORT/COMPARE: must use the locale for the sort/compare order
- MATCHES/MATCHFIELD: must determine if characters are numeric, alpha, etc via locale
- ICONV/OCONV: date, time and currency conversions all use the locale
- ALPHA, ISPRINT: properties must be based on the locale
- INPUT/PRINT: code page conversion to and from UTF-8

Normally, the LEN function returns the current byte length of the array, which is always kept up-to-date as the array increases or decreases in size. In international mode, the LEN function must return the *number of characters* rather than the *number of bytes* in the array. As a result, the array must be traversed in order to count the characters, causing a decrease in performance.

LOCATE usually compares strings directly, without regard for locale. In international mode, however, the locale is used during comparison. The same holds true for MATCHES, MATCHFIELD, SORT, COMPARE and property tests, since variables must first be converted to Unicode.

If international mode is enabled, conversion between code pages is required for Terminal I/O; however, this is a relatively slow operation. Whenever possible, it is ideal to use terminal emulators etc., which are capable of sending and receiving UTF-8, such that no code page conversion is necessary, thereby reducing the CPU overhead of conversion.

As all strings must be converted to UTF-8 encoding before compile time, and all read/write data is all presumed to be UTF-8 encoded, there should be no overhead to other functions, except as mentioned above or when functions are working on a character basis, e.g., substring extraction. If an account is not configured for international mode, the overhead is a simple bit test in a few functions.

## **Desktop Applications**

Desktop applications vary in their Unicode support, and as a result, have limited internationalization support:

- Limited Interfaces in Win98/ME
- VB 6 (Unicode is only supported on NT/Win2000)
- Microsoft Office 32 bit supports Unicode.
- 32-bit COM only supports Unicode.
- OLE DB, ODBC, ADO, RDO all COM components
- Java handles everything as Unicode

## **The Future and UTF-8**

The industry is converging on UTF-8 and Unicode for all internationalization. Microsoft NT is built on a base of Unicode; AIX, Sun, HP/UX all offer Unicode support. All the new web standards, such as HTML, XML, etc., support Unicode. The latest versions of Netscape Navigator and Internet Explorer both support Unicode. UNIX support for Unicode for directory names is provided via UTF-8.

Because of these difficulties, the major Unix distributors and developers foresee Unicode eventually replacing older legacy encodings, primarily in the UTF-8 form.

UTF-8 will more than likely be used exclusively in:

- Text files (source code, HTML files, email messages, etc.)
- File names
- Standard input and standard output, pipes
- Environment variables
- Cut and paste selection buffers
- Telnet, modem, and serial port connections to terminal emulators; and

- Any other places where byte sequences used to be interpreted in ASCII. For example, terminal emulators such as xterm or the Linux console driver transform every keystroke into the corresponding UTF-8 sequence and send it to the stdin of the foreground process.

## Summary

If it is certain that an application will only ever use ASCII characters, internationalization may be not necessary. However, with UTF-8 all ASCII characters stay the same. On the other hand, if providing an application to any additional markets is a possibility, you should carefully consider internationalization as a development process.

It is best to consider internationalization impacts early in the development of software products preferably before it is fully developed as significant application changes will be necessary, particularly if the product will be made available to the Asian market. Far more than simply translating literal strings, internationalization is a process that either can positively affect the quality and cost of development.

It is true that internationalization can lessen the performance of some important functions in the finished software product. However, if providing your application to a global marketplace it is an important business priority, tremendous carefully considering and understanding the process of internationalization will make gains in the development lifecycle and improved product quality.

## jBASE Environment Variables for Internationalization

Users can be configured for internationalization by setting the following environment variables.

### **JBASE\_I18N**

When the JBASE\_I18N environment variable is set, the application is expecting to execute in International mode.

**NOTE:** that the value of this environment variable can be modified by a LOGTO command. The value of the JBASE\_I18N variable will then be set according to the true or false value for the account.

### **JBASE\_CODEPAGE**

You can only set the JBASE\_CODEPAGE environment variable to a valid code page available with the ICU package. Use the jcodepages command for a list of currently available code pages. Conversion for input and output will only take place if configuring the account for international mode or the JBASE\_I18N variable is set.

### **JBASE\_LOCALE**

You can only set the JBASE\_LOCALE environment variable to a valid locale available with the ICU package. Use the jlocales command for a list of currently available locales. Only use the

configured locale if the account is configured for international mode or the JBASE\_I18N variable is set.

### **JBASE\_TIMEZONE**

You can only set the JBASE\_TIMEZONE environment variable to a valid time zone available with the ICU package. Use the `jtimezones` command for a list of currently available time zones. Only use the configured time zone if the account is configured for international mode or the JBASE\_I18N variable is set.

For example, the following environment variable configuration would configure a user for the French and the country locale specific for France and the code page set for latin1, iso-8859-1.

```
JBASE_I18N=1
JBASE_CODEPAGE=iso-8859-1
JBASE_LOCALE=fr_FR
```

## **Unicode and ICU Reference Information**

The following website contain useful information about the Unicode Standard and the IBM Open Source software package that has been implemented with the jBASE 4.1 release.

<http://www.unicode.org>

<http://oss.software.ibm.com/icu>

## **Locale Definition**

A Locale represents a specific geographical, political, or cultural region.

An operation that requires a Locale to perform its task is called *locale-sensitive* and uses the Locale to tailor information for the user. For example, displaying a date is a locale-sensitive operation: format the date according to the customs/conventions of the user's native country, region, or culture.

The first argument to the constructors is a valid **ISO Language Code**. These codes are the lower-case two-letter codes as defined by ISO-639.

<http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt>

The second argument to the constructors is a valid **ISO Country Code**. These codes are the upper-case two-letter codes as defined by ISO-3166.

[http://www.chemie.fu-berlin.de/diverse/doc/ISO\\_3166.html](http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html)

The third constructor requires a third argument: the **Variant**. Use the Variant codes to further define the locale, e.g., European countries now using the Euro use the variant code EURO.

## Configured Locales

LanguageCode\_CountryCode\_Variant (Available with ICU )

af	af_ZA ar	ar_AE	ar_BH	ar_DZ
ar_EG	ar_IQ	ar_JO	ar_KW	ar_LB
ar_LY	ar_MA	ar_OM	ar_QA	ar_SA
ar_SD	ar_SY	ar_TN	ar_YE	
be	be_BY	bg	bg_BG	
ca	ca_ES	ca_ES_EURO	cs	cs_CZ
da	da_DK	de	de_AT	de_AT_EURO
de_CH	de_DE	de_DE_EURO	de_LU	de_LU_EURO
el	el_GR	en	en_AU	en_BE
en_BW	en_CA	en_GB	en_IE	en_IE_EURO
en_NZ	en_SG	en_US	en_ZA	en_ZW
eo	es	es_AR	es_BO	es_CL
es_CO	es_CR	es_DO	es_EC	es_ES
es_ES_EURO	es_GT	es_HN	es_MX	es_NI
es_PA	es_PE	es_PR	es_PY	es_SV
es_US	es_UY	es_VE	et	et_EE

eu	eu_ES			
fa	fa_IR	fi	fi_FI	fi_FI_EURO
fo	fo_FO	fr	fr_BE	fr_BE_EURO
fr_CA	fr_CH	fr_FR	fr_FR_EURO	fr_LU
fr_LU_EURO				
ga	ga_IE	gl	gl_ES	gv
gv_GB				
he	he_IL	hi	hi_IN	hr
hr_HR	hu	hu_HU		
id	id_ID	is	is_IS	it
it_CH	it_IT	it_IT_EURO		
ja	ja_JP			
kl	kl_GL	ko	ko_KR	kok
kok_IN	kw	kw_GB		
lt	lt_LT	lv	lv_LV	
mk	mk_MK	mr	mr_IN	mt
mt_MT				
nl	nl_BE	nl_BE_EURO	nl_NL	nl_NL_EURO
no	no_NO	no_NO_B	no_NO_NY	
pl	pl_PL	pt	pt_BR	pt_PT
pt_PT_EURO				
ro	ro_RO	ru	ru_RU	ru_UA

sh	sh_YU	sk	sk_SK	sl
sl_SI	sq	sq_AL	sr	sr_YU
sv	sv_FI	sv_FI_AL	sv_SE	sw
sw_KE	sw_TZ			
ta	ta_IN	te	te_IN	th
th_TH	tr	tr_TR		
uk	uk_UA			
vi	vi_VN			
zh	zh_CN	zh_HK	zh_SG	zh_TW

### Language Codes (ISO-639)

aa	Afar	ab	Abkhazian	af	Afrikaans	am	Amharic
ar	Arabic	as	Assamese	ay	Aymara	az	Azerbaijani
ba	Bashkir	be	Byelorussian	bg	Bulgarian	bh	Bihari
bi	Bislama	bn	Bengali; Bangla	bo	Tibetan	br	Breton
ca	Catalan	co	Corsican	cs	Czech	cy	Welsh
da	Danish	de	German	dz	Bhutani	el	Greek
en	English	eo	Esperanto	es	Spanish	et	Estonian
eu	Basque	fa	Persian	fi	Finnish	fj	Fiji
fo	Faroese	fr	French	fy	Frisian	ga	Irish

gd Scots; Gaelic	gl Galician	gn Guarani	gu Gujarati
ha Hausa	he Hebrew (formerly iw)	hi Hindi	hr Croatian
Hu Hungarian	hy Armenian	ia Interlingua	id Indonesian (formerly in)
Ie Interlingue	ik Inupiak	is Icelandic	it Italian
iu Inuktitut	ja Japanese	jw Javanese	ka Georgian
kk Kazakh	kl Greenlandic	km Cambodian	kn Kannada
ko Korean	ks Kashmiri	ku Kurdish	ky Kirghiz
la Latin	ln Lingala	lo Laothian	lt Lithuanian
lv Latvian; Lettish	mg Malagasy	mi Maori	mk Macedonian
Ml Malayalam	mn Mongolian	mo Moldavian	mr Marathi
ms Malay	mt Maltese	my Burmese	na Nauru
ne Nepali	nl Dutch	no Norwegian	oc Occitan
om (Afan) Oromo	or Oriya	pa Punjabi	pl Polish
ps Pashto, Pushto	pt Portuguese	qu Quechua	rm Rhaeto- Romance
rn Kirundi	ro Romanian	ru Russian	rw Kinyarwanda
sa Sanskrit	sd Sindhi	sg Sangho	sh Serbo-Croatian
si Sinhalese	sk Slovak	sl Slovenian	sm Samoan

sn	Shona	so	Somali	sq	Albanian	sr	Serbian
ss	Siswati	st	Sesotho	su	Sundanese	sv	Swedish
sw	Swahili	ta	Tamil	te	Telugu	tg	Tajik
th	Thai	ti	Tigrinya	tk	Turkmen	tl	Tagalog
tn	Setswana	to	Tonga	tr	Turkish	ts	Tsonga
tt	Tatar	tw	Twi	ug	Uighur	uk	Ukrainian
ur	Urdu	uz	Uzbek	vi	Vietnamese	vo	Volapuk
wo	Wolof	xh	Xhosa	yi	Yiddish (formerly ji)	yo	Yoruba
za	Zhuang	zh	Chinese	zu	Zulu		

## Country Codes (ISO 3166)

Updated by the RIPE Network Coordination Centre, in coordination with the ISO 3166

Maintenance Agency, Berlin

Country	A 2	A 3	Number
AFGHANISTAN	AF	AFG	004
ALBANIA	AL	ALB	008
ALGERIA	DZ	DZA	012
AMERICAN SAMOA	AS	ASM	016
ANDORRA	AD	AND	020
ANGOLA	AO	AGO	024
ANGUILLA	AI	AIA	660
ANTARCTICA	AQ	ATA	010
ANTIGUA AND BARBUDA	AG	ATG	028
ARGENTINA	AR	ARG	032
ARMENIA	AM	ARM	051
ARUBA	AW	ABW	533
AUSTRALIA	AU	AUS	036
AUSTRIA	AT	AUT	040
AZERBAIJAN	AZ	AZE	031
BAHAMAS	BS	BHS	044
BAHRAIN	BH	BHR	048
BANGLADESH	BD	BGD	050
BARBADOS	BB	BRB	052

BELARUS	BY	BLR	112
BELGIUM	BE	BEL	056
BELIZE	BZ	BLZ	084
BENIN	BJ	BEN	204
BERMUDA	BM	BMU	060
BHUTAN	BT	BTN	064
BOLIVIA	BO	BOL	068
BOSNIA AND HERZEGOWINA	BA	BIH	070
BOTSWANA	BW	BWA	072
BOUVET ISLAND	BV	BVT	074
BRAZIL	BR	BRA	076
BRITISH INDIAN OCEAN TERRITORY	IO	IOT	086
BRUNEI DARUSSALAM	BN	BRN	096
BULGARIA	BG	BGR	100
BURKINA FASO	BF	BFA	854
BURUNDI	BI	BDI	108
CAMBODIA	KH	KHM	116
CAMEROON	CM	CMR	120
CANADA	CA	CAN	124
CAPE VERDE	CV	CPV	132
CAYMAN ISLANDS	KY	CYM	136
CENTRAL AFRICAN REPUBLIC	CF	CAF	140

CHAD	TD	TCD	148
CHILE	CL	CHL	152
CHINA	CN	CHN	156
CHRISTMAS ISLAND	CX	CXR	162
COCOS (KEELING) ISLANDS	CC	CCK	166
COLOMBIA	CO	COL	170
COMOROS	KM	COM	174
CONGO	CG	COG	178
COOK ISLANDS	CK	COK	184
COSTA RICA	CR	CRI	188
COTE D'IVOIRE	CI	CIV	384
CROATIA (local name: Hrvatska)	HR	HRV	191
CUBA	CU	CUB	192
CYPRUS	CY	CYP	196
CZECH REPUBLIC	CZ	CZE	203
DENMARK	DK	DNK	208
DJIBOUTI	DJ	DJI	262
DOMINICA	DM	DMA	212
DOMINICAN REPUBLIC	DO	DOM	214
EAST TIMOR	TP	TMP	626
ECUADOR	EC	ECU	218
EGYPT	EG	EGY	818

EL SALVADOR	SV	SLV	222
EQUATORIAL GUINEA	GQ	GNQ	226
ERITREA	ER	ERI	232
ESTONIA	EE	EST	233
ETHIOPIA	ET	ETH	231
FALKLAND ISLANDS (MALVINAS)	FK	FLK	238
FAROE ISLANDS	FO	FRO	234
FIJI	FJ	FJI	242
FINLAND	FI	FIN	246
FRANCE	FR	FRA	250
FRANCE, METROPOLITAN	FX	FXX	249
FRENCH GUIANA	GF	GUF	254
FRENCH POLYNESIA	PF	PYF	258
FRENCH SOUTHERN TERRITORIES	TF	ATF	260
GABON	GA	GAB	266
GAMBIA	GM	GMB	270
GEORGIA	GE	GEO	268
GERMANY	DE	DEU	276
GHANA	GH	GHA	288
GIBRALTAR	GI	GIB	292
GREECE	GR	GRC	300
GREENLAND	GL	GRL	304

GRENADA	GD	GRD	308
GUADELOUPE	GP	GLP	312
GUAM	GU	GUM	316
GUATEMALA	GT	GTM	320
GUINEA	GN	GIN	324
GUINEA-BISSAU	GW	GNB	624
GUYANA	GY	GUY	328
HAITI	HT	HTI	332
HEARD AND MC DONALD ISLANDS	HM	HMD	334
HONDURAS	HN	HND	340
HONG KONG	HK	HKG	344
HUNGARY	HU	HUN	348
ICELAND	IS	ISL	352
INDIA	IN	IND	356
INDONESIA	ID	IDN	360
IRAN (ISLAMIC REPUBLIC OF)	IR	IRN	364
IRAQ	IQ	IRQ	368
IRELAND	IE	IRL	372
ISRAEL	IL	ISR	376
ITALY	IT	ITA	380
JAMAICA	JM	JAM	388
JAPAN	JP	JPN	392

JORDAN	JO	JOR	400
KAZAKHSTAN	KZ	KAZ	398
KENYA	KE	KEN	404
KIRIBATI	KI	KIR	296
KOREA, DEMOCRATIC PEOPLE'S REPUBLIC OF	KP	PRK	408
KOREA, REPUBLIC OF	KR	KOR	410
KUWAIT	KW	KWT	414
KYRGYZSTAN	KG	KGZ	417
LAO PEOPLE'S DEMOCRATIC REPUBLIC	LA	LAO	418
LATVIA	LV	LVA	428
LEBANON	LB	LBN	422
LESOTHO	LS	LSO	426
LIBERIA	LR	LBR	430
LIBYAN ARAB JAMAHIRIYA	LY	LBY	434
LIECHTENSTEIN	LI	LIE	438
LITHUANIA	LT	LTU	440
LUXEMBOURG	LU	LUX	442
MACAU	MO	MAC	446
MACEDONIA, THE FORMER YUGOSLAV REPUBLIC OF	MK	MKD	807
MADAGASCAR	MG	MDG	450
MALAWI	MW	MWI	454

MALAYSIA	MY	MYS	458
MALDIVES	MV	MDV	462
MALI	ML	MLI	466
MALTA	MT	MLT	470
MARSHALL ISLANDS	MH	MHL	584
MARTINIQUE	MQ	MTQ	474
MAURITANIA	MR	MRT	478
MAURITIUS	MU	MUS	480
MAYOTTE	YT	MYT	175
MEXICO	MX	MEX	484
MICRONESIA, FEDERATED STATES OF	FM	FSM	583
MOLDOVA, REPUBLIC OF	MD	MDA	498
MONACO	MC	MCO	492
MONGOLIA	MN	MNG	496
MONTSERRAT	MS	MSR	500
MOROCCO	MA	MAR	504
MOZAMBIQUE	MZ	MOZ	508
MYANMAR	MM	MMR	104
NAMIBIA	NA	NAM	516
NAURU	NR	NRU	520
NEPAL	NP	NPL	524
NETHERLANDS	NL	NLD	528

NETHERLANDS ANTILLES	AN	ANT	530
NEW CALEDONIA	NC	NCL	540
NEW ZEALAND	NZ	NZL	554
NICARAGUA	NI	NIC	558
NIGER	NE	NER	562
NIGERIA	NG	NGA	566
NIUE	NU	NIU	570
NORFOLK ISLAND	NF	NFK	574
NORTHERN MARIANA ISLANDS	MP	MNP	580
NORWAY	NO	NOR	578
OMAN	OM	OMN	512
PAKISTAN	PK	PAK	586
PALAU	PW	PLW	585
PANAMA	PA	PAN	591
PAPUA NEW GUINEA	PG	PNG	598
PARAGUAY	PY	PRY	600
PERU	PE	PER	604
PHILIPPINES	PH	PHL	608
PITCAIRN	PN	PCN	612
POLAND	PL	POL	616
PORTUGAL	PT	PRT	620
PUERTO RICO	PR	PRI	630

QATAR	QA	QAT	634
REUNION	RE	REU	638
ROMANIA	RO	ROM	642
RUSSIAN FEDERATION	RU	RUS	643
RWANDA	RW	RWA	646
SAINT KITTS AND NEVIS	KN	KNA	659
SAINT LUCIA	LC	LCA	662
SAINT VINCENT AND THE GRENADINES	VC	VCT	670
SAMOA	WS	WSM	882
SAN MARINO	SM	SMR	674
SAO TOME AND PRINCIPE	ST	STP	678
SAUDI ARABIA	SA	SAU	682
SENEGAL	SN	SEN	686
SEYCHELLES	SC	SYC	690
SIERRA LEONE	SL	SLE	694
SINGAPORE	SG	SGP	702
SLOVAKIA (Slovak Republic)	SK	SVK	703
SLOVENIA	SI	SVN	705
SOLOMON ISLANDS	SB	SLB	090
SOMALIA	SO	SOM	706
SOUTH AFRICA	ZA	ZAF	710
SOUTH GEORGIA AND THE SOUTH SANDWICH ISLANDS	GS	SGS	239

SPAIN	ES	ESP	724
SRI LANKA	LK	LKA	144
ST. HELENA	SH	SHN	654
ST. PIERRE AND MIQUELON	PM	SPM	666
SUDAN	SD	SDN	736
SURINAME	SR	SUR	740
SVALBARD AND JAN MAYEN ISLANDS	SJ	SJM	744
SWAZILAND	SZ	SWZ	748
SWEDEN	SE	SWE	752
SWITZERLAND	CH	CHE	756
SYRIAN ARAB REPUBLIC	SY	SYR	760
TAIWAN	TW	TWN	158
TAJIKISTAN	TJ	TJK	762
TANZANIA, UNITED REPUBLIC OF	TZ	TZA	834
THAILAND	TH	THA	764
TOGO	TG	TGO	768
TOKELAU	TK	TKL	772
TONGA	TO	TON	776
TRINIDAD AND TOBAGO	TT	TTO	780
TUNISIA	TN	TUN	788
TURKEY	TR	TUR	792
TURKMENISTAN	TM	TKM	795

TURKS AND CAICOS ISLANDS	TC	TCA	796
TUVALU	TV	TUV	798
UGANDA	UG	UGA	800
UKRAINE	UA	UKR	804
UNITED ARAB EMIRATES	AE	ARE	784
UNITED KINGDOM	GB	GBR	826
UNITED STATES	US	USA	840
UNITED STATES MINOR OUTLYING ISLANDS	UM	UMI	581
URUGUAY	UY	URY	858
UZBEKISTAN	UZ	UZB	860
VANUATU	VU	VUT	548
VATICAN CITY STATE (HOLY SEE)	VA	VAT	336
VENEZUELA	VE	VEN	862
VIET NAM	VN	VNM	704
VIRGIN ISLANDS (BRITISH)	VG	VGB	092
VIRGIN ISLANDS (U.S.)	VI	VIR	850
WALLIS AND FUTUNA ISLANDS	WF	WLF	876
WESTERN SAHARA	EH	ESH	732
YEMEN	YE	YEM	887
YUGOSLAVIA	YU	YUG	891
ZAIRE	ZR	ZAR	180
ZAMBIA	ZM	ZMB	894

## Configured Code Pages and Aliases

When internationalizing applications it is necessary to ensure that data is correctly represented in the expected character set for the end user. This requirement is usually dependent upon the code page capabilities of the input and output devices used. Eventually this should move more and more to communicating solely in UTF-8, however for applications that deal with other character sets, this means that data must be converted to and from UTF-8.

The ICU library package provides a comprehensive character set conversion framework, mapping tables, and implementations for many encodings, including Unicode encodings. These mapping tables have mostly originated from the IBM code page repository. However, for non-IBM code pages there is usually an equivalent code configured. However, the textual data format is generic, and data for other code page mapping tables can be added. There is no single, authoritative source of precise definitions of many of the encodings and their names. However, IANA is the best source for names, and the Character Set repository provided by ICU is a good source of encoding definitions for each platform.

## UTF and ASCII Code Pages

### Code Page : utf-8

Aliases : cp1208, ibm-1208, utf-8, UTF8

### Code Page : utf-16be

Aliases : utf-16be, UTF16\_BigEndian

### Code Page : utf-16le

Aliases : utf-16le, UTF16\_LittleEndian

### Code Page : ISO-10646-UCS-2

Aliases : utf-16, ucs-2, cp1200, ibm-1200, utf-16, csUnicode, ISO-10646-UCS-2

### Code Page : utf-32be

Aliases : utf-32be, UTF32\_BigEndian

### Code Page : utf-32le

Aliases : utf-32le, UTF32\_LittleEndian

### Code Page : ISO-10646-UCS-4

Aliases : utf-32, ucs-4, utf-32, csUCS4, ISO-10646-UCS-4

**Code Page : ANSI\_X3.4-1968**

Aliases : us-ascii, iso-ir-6, 646, csASCII, us, iso646-us, ISO\_646.irv:1991, ANSI\_X3.4-1986, ANSI\_X3.4-1968, US-ASCII, ascii-7, ascii, us-ascii, ibm-367

## Latin Code Pages

**Code Page : ISO\_8859-1:1987**

Aliases : iso-8859-1, ANSI\_X3.110-1983, 11, ISO\_8859-1:1987, cp367, iso-ir-100, csisolatin1, 8859-1, latin1, cp819, ibm-819, iso-8859-1, LATIN\_1

**Code Page : ISO\_8859-2:1987**

Aliases : iso-8859-2, 12, ISO\_8859-2:1987, iso-ir-101, csisolatin2, 8859-2, latin2, cp912, iso-8859-2, ibm-912

**Code Page : ISO\_8859-3:1988**

Aliases : iso-8859-3, 13, ISO\_8859-3:1988, iso-ir-109, csisolatin3, 8859-3, cp913, latin3, iso-8859-3, ibm-913

**Code Page : ISO\_8859-4:1988**

Aliases : iso-8859-4, 14, ISO\_8859-4:1988, iso-ir-110, csisolatin4, 8859-4, cp914, latin4, iso-8859-4, ibm-914

**Code Page : ISO\_8859-5:1988**

Aliases : iso-8859-5, ISO\_8859-5:1988, iso-ir-144, csisolatincyrillic, 8859-5, cp915, cyrillic, iso-8859-5, ibm-915

**Code Page : ISO\_8859-6:1987**

Aliases : iso-8859-6, asmo-708, ecma-114, ISO\_8859-6:1987, iso-ir-127, csisolatinarabic, 8859-6, cp1089, arabic, iso-8859-6, ibm-1089

**Code Page : ISO\_8859-7:1987**

Aliases : iso-8859-7, ISO\_8859-7:1987, iso-ir-126, csisolatingreek, 8859-7, ecma-118, elot\_928, greek8, greek, iso-8859-7, cp813, ibm-4909

**Code Page : ISO\_8859-8:1988**

Aliases : iso-8859-8, ISO\_8859-8:1988, iso-ir-138,  
csisolatinhebrew, 8859-8, cp916, hebrew, iso-8859-8, ibm-916

**Code Page : ISO\_8859-9:1989**

Aliases : iso-8859-9, 15, ISO\_8859-9:1989, iso-ir-148,  
csisolatin5, 8859-9, cp920, latin5, ECMA-128, iso-8859-9, ibm-920

**Code Page : iso-8859-15**

Aliases : csisolatin9, csisolatin0, latin0, 8859-15, cp923,  
latin9, iso-8859-15, ibm-923

## Mircosoft Windows Code Pages

**Code Page : windows-1250**

Aliases : windows-1250, cp1250, windows-1250, ibm-5346

**Code Page : windows-1251**

Aliases : windows-1251, cp1251, windows-1251, ibm-5347

**Code Page : windows-1252**

Aliases : windows-1252, cp1252, windows-1252, ibm-5348

**Code Page : windows-1253**

Aliases : windows-1253, cp1253, windows-1253, ibm-5349

**Code Page : windows-1254**

Aliases : windows-1254, cp1254, windows-1254, ibm-5350

**Code Page : windows-1255**

Aliases : windows-1255, cp1255, windows-1255, ibm-5351

**Code Page : windows-1256**

Aliases : windows-1256, cp1256, windows-1256, ibm-5352

**Code Page : windows-1257**

Aliases : windows-1257, cp1257, windows-1257, ibm-5353

**Code Page : windows-1258**

Aliases : windows-1258, cp1258, windows-1258, ibm-5354

## Japanese, Chinese, Korean Code Pages

**Code Page : ISO-2022**

Aliases : cp2022, 2022, ISO-2022, ISO\_2022

**Code Page : ISO-2022-JP**

Aliases : ISO-2022-JP, csISO2022JP, ISO-2022-JP,  
ISO\_2022,locale=ja,version=0

**Code Page : ISO-2022-KR**

Aliases : ISO-2022-KR, csISO2022KR, ISO-2022-KR,  
ISO\_2022,locale=ko

**Code Page : ISO-2022-CN**

Aliases : csISO2022CN, ISO-2022-CN, ISO\_2022,locale=zh,version=0

**Code Page : Shift\_JIS**

Aliases : x-sjis, windows-31j, csshiftjis, ms\_kanji, cp932, cp943,  
sjis, csWindows31J, Shift\_JIS, ibm-943

**Code Page : Big5**

Aliases : cp950, x-big5, csBig5, Big5, ibm-1370

**Code Page : GB\_2312-80**

Aliases : GB2312, zh\_cn, cp936, gb2312-1980, GB2312, gb, chinese,  
gbk, csISO58GB231280, iso-ir-58, GB\_2312-80, ibm-1386

**Code Page : EUC-JP**

Aliases : X-EUC-JP, extended\_unix\_code\_packed\_format\_for\_japanese,  
eucjis, ibm-eucJP, EUC-JP, ibm-33722

**Code Page : EUC-KR**

Aliases : EUC-KR, csEUCKR, ibm-eucKR, EUC-KR, ibm-970

**Code Page : EUC-TW**

Aliases : cns11643, ibm-eucTW, EUC-TW, ibm-964

**Code Page : EUC-CN**

Aliases : ibm-eucCN, EUC-CN, ibm-1383

**Code Page : KOI8-R**

Aliases : KOI8-R, cskoi8r, koi8, cp878, KOI8-R, ibm-878

**Code Page : korean**

Aliases : ksc, cp949, cp1363, ibm-1363

## Comment Sheet

Please give page number and description for any errors found:

Page	Error

Please use the box below to describe any material you think is missing; describe any material which is not easily understood; enter any suggestions for improvement; provide any specific examples of how you use your system which you think would be useful to readers of this manual. Continue on a separate sheet if necessary.

Copy and paste this page to a word document and include your name address and telephone number. Email to [documentation@temenos.com](mailto:documentation@temenos.com)