



Unicode in GUPTA® SQLBase® International

By Kevin Watts
Senior Technical Support Engineer

June, 2006

GUPTA™

Abstract	2
Introduction	2
I Don't Want To Change Anything!	2
Unicode in SQLBase	3
<i>Additional Data Types</i>	3
<i>Programmatic interfaces</i>	4
<i>New function names ending in 2:</i>	4
<i>New Function names ending in A or W:</i>	4
<i>Data Type Conversion</i>	5
<i>New Scalar Functions of Interest</i>	5
<i>Scalar Functions Modified to Accept Unicode Expressions</i>	5
SQLTalk for SQLBase International	6
Conclusion	6
Appendix A	7





SQLBase International provides you with international business solutions.

Applications can run exactly as they do with prior versions.

Abstract

This paper will introduce you to the new internationalization features of SQLBase International with an emphasis on our Unicode implementation.

Introduction

Unicode is one facet of GUPTA's strategy to improve the "internationalization" of SQLBase applications. As companies integrate e-commerce on a global scale, they can increase revenues, decrease expenses, and improve customer communications with proper internationalization strategies. A company that chooses appropriate products that can hide cultural nuances and the technical complexities of locale-specific software requirements has a distinct advantage over one that must bear the high cost and large effort of building internal solutions.

SQLBase NLS (National Language Support) had already provided extensive database customization to address locale-specific problems, but it had two notable weaknesses. NLS definitions are specific to the entire database server. It was not possible to mix different NLS databases on a single server. NLS definitions were also specific to a client, making it impossible for a single client to communicate with multiple database servers with different locale orientations.

SQLBase now recognizes Unicode characters beyond the ASCII character set, and they can be used to form data and object names.

Closely related to Unicode is the concept of collations. A collation is a set of rules that define how characters are to be compared with each other. SQLBase International supports a binary collation. If you want sorting to behave as it has in the past, no action is required. However, SQLBase International offers a multitude of additional collations.

"Using Collations with SQLBase International" is a white paper that defines and explores in detail the various collations that are now available to you.

I Don't Want To Change Anything!

Let's allay some fears right away. Many of you want to migrate your databases to SQLBase International and have everything run exactly as they do with your current version. You might be concerned you will have to make programming or script changes in order to migrate because SQLBase is now Unicode compliant. Don't worry! If you don't currently use National Language Support (NLS) or specifically utilize the new Unicode features of SQLBase International, everything from a client perspective will run exactly as it did under prior versions.

For example, if you unload the ISLAND sample database from SQLBase 9.0.1 and load it into SQLBase International, you need do absolutely nothing in order to run the sample applications exactly as before.

If you do use NLS you will have to make some changes. There is an excellent white paper named "SQLBase NLS Migration to Unicode" that will help get you started.



What Is Unicode?

The Unicode Standard is a character coding system designed to support the worldwide interchange, display, and processing of the written texts of the diverse languages of the world. It also supports classical written texts of many historical written languages.

The Unicode Consortium declares, "Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language."

“Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language.”

The main input interface to your computer, your keyboard, was modeled on mechanical typewriters. Since much of the initial computer development was done in the United States, keyboards were patterned on typical typewriter keyboards in use for English in the United States.

Individual keys were mapped to numerical values and stored in single 8-bit bytes as what came to be known as the ASCII character set. Valid values were 0 to 127. The ASCII character set soon was extended to 255 characters to accommodate extra characters that were needed in various European character sets. As the popularity of computing expanded into Asian markets, even the expanded set was not enough to hold all of the characters. This is when multi-byte character sets were introduced. They consist of 2-byte combinations that represent a single character.

Character set proliferation was typically not a problem as long as all work was being done within a single set. But big problems began to emerge when interoperability among various character sets was required. Consider a single byte, for example 0xE6. If you are using a Windows Western character set, the code represents "æ" However, 0xE6 in the Windows Central European character set represents "ç" (c with an acute). How do we interpret 0xE6 when we encounter it in multi-set operations? This is the type of problem Unicode was developed to solve. Unicode guarantees a unique number is assigned to every member of multiple character sets.

Unicode can be thought of as a super character set that includes character sets from around the world.

Providing a comprehensive description of Unicode is well beyond the scope of this paper. We are going to concentrate on how Unicode has been implemented in SQLBase. If you wish more comprehensive information, excellent sources can be found at <http://www.unicode.org>.

Unicode in SQLBase

SQLBase has several additional features to support the Unicode standard. We will introduce the most important ones in this paper.

Additional Data Types

New data types were added that affect the SQLBase DDL SQL language support. The SQL-99 Standard specifies the use of NCHAR and NVARCHAR for the use of a pre-defined Unicode character set. We have added both of these data types to SQLBase.

```
CREATE TABLE UNICODE_TEST(COL_UNICODE NCHAR(50))
```

SQLBase International has many new features to support Unicode.



```
CREATE TABLE UNICODE_TEST(COL_UNICODE NVARCHAR(50))
```

Programmatic interfaces

SQLBase programming interfaces were changed to support the new SQL language features and to allow interoperability between the old and new data types.

New External Data Types:

- SQLENCH -- NATIONAL CHARACTER
- SQLENCV -- NATIONAL VARCHAR
- SQLELNC -- LONG NATIONAL CHARACTER

New Internal Data Types:

- SQLDNCH -- National Character
- SQLDLNC -- Long National Character

New Program Data Types:

- SQLPNCH -- National character
- SQLPLNC -- Long national character

New function names ending in 2:

In some listings in our documents, you will see two function names together, such as sqlbln and sqlbln2. In such cases, the two functions are designed for the same purpose. However, the first name is the one used in versions of SQLBase prior to 10.0. The name ending in "2" was introduced in version 10.0 to deal with the new, greater capacities in the database.

If your particular database does not use these new features, you can continue to use older function names in your application programs. If you do adopt these new features, you must move to the version of the function whose name ends in "2".

New Function names ending in A or W:

Some functions must be compatible with the newer sizes in version 10.0 and also with smaller sizes in versions prior to 10.0. Such functions now have a base function name that is the same as it was in versions prior to 10.0, plus two new functions, one with "A" appended to the name, and one with "W" appended to the name. For example, function sqlbdb (backup database) is still present in the SQLBase API, but it is now mapped to either function sqlbdbA or sqlbdbW. The mapping is determined by a #define directive, like so:

```
#ifdef _UNICODE
#define sqlbdb sqlbdbW
#else
#define sqlbdb sqlbdbA
#endif
```

When _UNICODE is defined, it indicates that the client is a Unicode (Wide character) client. The "W" version of the function will be used, and data type SQLTDAPW will be used for certain parameters. If

SQLBase has been extended to support Unicode.

New data types and functions give you access to Unicode features.



SQLBase 10 continues its proud tradition of offering the most flexible data type conversions in the industry.

New and modified scalar functions give you a rich set of Unicode string manipulation tools.

_UNICODE is not defined, it indicates that the client uses ANSI characters. The "A" version of the function will be used, and data type SQLTDAP will be used for those parameters. Under normal circumstances, you need never call the "A" or "W" version of the functions directly. Simply manage the _UNICODE define properly and always call the base name of the functions.

See the SQLBase SQL Application Programming Interface Reference for complete details.

Data Type Conversion

Implicit conversions are made when CHAR and NCHAR operations are performed. Expressions comprised of a CHAR and an NCHAR expression results in an NCHAR expression. For example, concatenating a CHAR with an NCHAR will result in an NCHAR.

```
SELECT COL_CHAR || COL_NCHAR FROM TABLE_UNICODE
```

Scalar functions have been added to allow conversion between CHAR and NCHAR data types.

New Scalar Functions of Interest

- @NCHAR - Returns a Unicode character from a given decimal code
- @NCODE - Returns the decimal code from the first character in a Unicode string
- @NSTRING - Converts a number to a Unicode string.
- @NSORTKEY - Converts a Unicode string to a machine-readable format that allows faster comparisons. Note: A binary comparison of two @NSORTKEY items yield better performance than a comparison of two Unicode strings.

Scalar Functions Modified to Accept Unicode Expressions

```
@CHOOSE
@COALESCE
@DATEVALUE
@DECIMAL
@DECODE
@EXACT
@FIND
@ISNA
@LEFT
@LENGTH
@LOWER
@NULLVALUE
@PROPER
@REPEAT
@REPLACE
@RIGHT
@SCAN
@STRING
@SUBSTRING
@TRIM
@UPPER
@VALUE
```



SQLTalk for SQLBase International

SQLTalk, the primary DBA interface to SQLBase, has also been modified to support Unicode. See Appendix A for a script showing you some of the things you can do when you use the Unicode features of SQLBase International.

SQLBase 10 will help you enhance your products to compete in global markets.

Conclusion

With SQLBase International, you have the tools you need to enhance global information dissemination to your customers and partners. The bit of an education curve involved in learning how to use these features is well worth the effort. Explore the new features of SQLBase International in the documentation and in these white papers and start leveraging your products on the worldwide stage.

Kommentar [cm1]: More descriptions of each of the tasks need to be inserted so that each operation is explained. You can use the BreakOut boxes if you like.
Also the resulting data tables need to be put in to "Word" tables. This will reduce formatting problems going forward with the document.



Appendix A

Sample SQLTalk script.

```
SELECT * FROM DAYS_OF_WEEK ORDER BY OFFSET
```

```

/
FOREIGN_LOCALE      FOREIGN_NAME      OFFSET
de                  Sonntag           0
en                  Sunday            0
it                  domenica          0
fr                  dimanche          0
it                  lunedì            1
fr                  lundi              1
en                  Monday            1
de                  Montag            1
de                  Dienstag         2
fr                  mardi             2
en                  Tuesday           2
it                  martedì           2
de                  Mittwoch          3
en                  Wednesday         3
it                  mercoledì         3
fr                  mercredi          3
de                  Donnerstag        4
en                  Thursday          4
it                  giovedì           4
fr                  Jeudi             4
it                  venerdì           5
de                  Freitag           5
fr                  vendredi          5
en                  Friday            5
de                  Samstag           6
fr                  samedi            6
en                  Saturday          6
it                  sabato            6

```

28 ROWS SELECTED

```
SELECT * FROM DAYS_OF_WEEK WHERE FOREIGN_LOCALE='en' ORDER BY OFFSET
```

```

/
FOREIGN_LOCALE      FOREIGN_NAME      OFFSET
en                  Sunday            0
en                  Monday            1
en                  Tuesday           2
en                  Wednesday         3
en                  Thursday          4
en                  Friday            5
en                  Saturday          6

```

7 ROWS SELECTED

```
SELECT * FROM DAYS_OF_WEEK WHERE FOREIGN_LOCALE='de' ORDER BY OFFSET
```

```

/
FOREIGN_LOCALE      FOREIGN_NAME      OFFSET
de                  Sonntag           0
de                  Montag            1
de                  Dienstag         2
de                  Mittwoch          3
de                  Donnerstag        4
de                  Freitag           5
de                  Samstag           6

```

7 ROWS SELECTED

```
SELECT * FROM DAYS_OF_WEEK WHERE FOREIGN_LOCALE='fr' ORDER BY OFFSET
```

```

/
FOREIGN_LOCALE      FOREIGN_NAME      OFFSET
fr                  dimanche          0
fr                  Lundi             1
fr                  mardi             2

```

```

fr          mercredi          3
fr          jeudi             4
fr          vendredi          5
fr          Samedi            6

```

7 ROWS SELECTED

```

SELECT * FROM DAYS_OF_WEEK WHERE FOREIGN_LOCALE='it' ORDER BY OFFSET

```

```

/
FOREIGN_LOCALE  FOREIGN_NAME  OFFSET
it              domenica    0
it              Lunedì      1
it              martedì     2
it              mercoledì   3
it              giovedì     4
it              venerdì     5
it              sabato      6

```

7 ROWS SELECTED

```

SELECT * FROM DAYS_OF_WEEK2 ORDER BY OFFSET

```

```

/
OFFSET  DE_NAME  FR_NAME  IT_NAME  EN_NAME
0  Sonntag  Dimanche  Domenica  Sunday
1  Montag   Lundi    Lunedì   Monday
2  Dienstag  Mardi   martedì  Tuesday
3  Mittwoch  Mercredi Mercoledì Wednesday
4  Donnerstag  Jeudi   giovedì  Thursday
5  Freitag   Vendredi venerdì   Friday
6  Samstag   Samedi  Sabato   Saturday

```

7 ROWS SELECTED

```

REMARK
\
New System Tables
/

```

```

ROWCOUNT SYSCOLLATIONS
/

```

```

254 ROWS IN TABLE

```

```

SET LIMIT 10
/

```

```

SELECT * FROM SYSCOLLATIONS
/

```

```

ID  NAME
0  BINARY
1  UCA_CS_AS
2  UCA_CS_AI
3  UCA_CI_AS
4  UCA_CI_AI
5  German_phonebook_CS_AS
6  German_phonebook_CS_AI
7  German_phonebook_CI_AS
8  German_phonebook_CI_AI
9  Spanish_traditional_CS_AS

```

10 ROWS SELECTED

```

ROWCOUNT SYSCHARACTERSETS
/

```

```

197 ROWS IN TABLE

```

```

SELECT ID, NAME FROM SYSCHARACTERSETS
/

```

```

ID  NAME
0
1  UTF-8
2  UTF-16
3  UTF-16BE
4  UTF-16LE
5  UTF-32

```

Sample SQLTalk script.

```
6 UTF-32BE
7 UTF-32LE
8 UTF-7
9 SCSU
```

```
10 ROWS SELECTED
```

```
ALTER TABLE LATIN_GERMAN COLLATE DEFAULT
/
SET LIMIT OFF
/
SELECT * FROM LATIN_GERMAN ORDER BY LATIN_LETTER
/
LATIN_LETTER
=====
```

```
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
Ä
Ö
Û
ß
ä
```

Sample SQLTalk script.

ö
ü

59 ROWS SELECTED

REMARK

\
Change the collation to German Phonebook with both case sensitivity and
accent sensitivity activate. Notice the difference in output.

/

ALTER TABLE LATIN_GERMAN COLLATE GERMAN_PHONEBOOK_CS_AS

/

SELECT * FROM LATIN_GERMAN ORDER BY LATIN_LETTER

/

LATIN_LETTER

=====

A

a

Ä

ä

B

b

C

c

D

d

E

e

F

f

G

g

H

h

I

i

J

j

K

k

L

l

M

m

N

n

O

o

Ö

ö

P

p

Q

q

R

r

S

s

ß

T

t

U

u

Ü

ü

V

v

W

w

X

x

Y

Sample SQLTalk script.



y
z
z

59 ROWS SELECTED

The End.