Transbase®

WHITE PAPER

How to Convert Existing Transbase® Databases into NLS Databases

1. General Remarks

For historical reasons, character data was stored in arrays of bytes with byte values ranging from 0 to 0x7f (note that we use hexadecimal representation throughout this paper to denote byte values). The so-called 0-byte frequently was used to define the end of a character string and was not permitted to be part of a string. On PCs and UNIX-based systems, the 7-bit ASCII code was typically used to define the interpretation of bytes. E.g. the letter "capital X" was denoted by the byte with value 0x58, while the letter "3" was denoted by 0x33. Byte values higher than 0x7f did not appear, originally the high-order bit was used to denote a so-called parity bit in order to detect eventual transmission errors.

ASCII did not cope with non-US letters such as a German ä or a Spanish ç. First some coding systems used ASCII characters like '[' to redefine them as 'Ä' in Germany. Later, various coding systems were defined who used the full 8-bit-range, i.e. all values from 0 to 0xff (making the use of parity bits obsolete). Unfortunately, no standards were available so that at least the following coding systems have gained some importance:

- The codepage 437 was defined by IBM-PC and still is being used in MSDOS windows. In this codepage the German 'Ä' is denoted by 0x8E.
- ISO 8859 defined several variants for single-byte codes: 8859-1 (sometime called Latin-1) covers all characters used in Western European languages (mapping the German 'Ä' to 0xC4), while 8859-2 covers Eastern European languages, 8859-7 covers Greek characters (e.g. mapping the Greek capital D ('D') to 0xC4, 8859-8 covers Hebrew characters in the upper half. Note that the lower half (bytes from 0x00 to 0x7f) always is identical to ASCII.

A byte value of 0xC4 thus could be interpreted either as German 'Ä' in ISO-8859-1 or as Greek 'D' in ISO-8859-7. In particular, there was no way to store both characters in the same database.
Even worse, a database server could be connected to a German client and a Greek client; data inserted from the German client could not be interpreted by the Greek client and vice versa.

The definition of UNICODE (ISO 10646) finally solved this problem by assigning each character in the world a unique code (namely its UNICODE). Since more than 256 different characters had to be coded, UNICODE uses 2 (UCS-2) or 4 (UCS-4) bytes for each character. We use the notation U+C4. A German 'Ä' e.g. is assigned the unicode U+C4 while the Greek 'D' is assigned the unicode U+394. UNICODE in particular covers East-Asian characters like Chinese Kanji as well as Indian and Arabic characters, too.

Spending four bytes per character, however, seems to be fairly inefficient for Western languages. Furthermore, character processing is usually done byte by byte in computer programs which would require a complete redesign to adapt them for UNICODE strings. Fortunately, together with UNICODE also a "Unicode transformation format" (UTF) was introduced which maps Unicode characters into byte (UTF-8) or word sequences (UTF-16). UTF-8 therefore is the most suitable format to store and process UNICODE strings in computer programs today. In particular, ASCII characters (from U+0 to U+7F) are mapped one-to-one into UTF-8, characters between U+80 and U+7FF are mapped into 2-byte sequences, characters between U+800 and U+FFFF are mapped into 3-byte sequences and so on. The longest possible sequence takes six bytes. ASCII characters do not take more space than before; programs that deal only with ASCII characters need not be modified at all.

Prior to version 5.3 of Transbase®, character data was stored simply as arrays of bytes. Transbase® left the interpretion of bytes to application programs. Although it would have been possible to store multi-byte-strings, there would have been misbehaviour e.g. when a LIKE predicate would have to match one character (that actually took two or more bytes).

Since version 5.3, Transbase® supports an explicit database coding. The available codings are:

- **ASCII:** Only characters in the range 0x0 to 0x7f are valid.
- **Single Byte:** Characters in the range 0x0 to 0xff are permitted; the mapping into UNICODE is defined by a "locale" setting. Character strings are processed byte by byte (which is character by character).
- **UTF-8:** Only valid UTF-8 strings are permitted. Character strings are processed character by character (with respect to UTF-8 properties).
- **EUC and S/JIS:** Like UTF-8, strings are processed with respect to their coding as byte sequences.
- **Propriet:** This coding is provided for migration from and compatibility to former versions of Transbase®.

After this lengthy introduction, the following paragraphs describe how existing databases can be migrated to newer versions of Transbase®.

We have to distinguish the following cases:

- Interpretation of characters is not essential: Simply continue to use "Propriet" as coding.
- Database contents are in one of the codings supported; just enter this coding into the database's configuration file and use the database as before.
- Database contents shall be converted into a true NLS format, in particular: UTF-8.

By nature, the migration support of Transbase® is limited; actually, old databases did not know about codepages and therefore do not provide any support upon creating archives. They can simply put their data "as is".

However, Transbase® versions newer than 5.3 know about codepages and they can particularily read spool files from old databases with an extended spool statement:

```
SPOOL <table> FROM <file> CODEPAGE <codepage>
```

Where `<codepage>` is one of UTF8, UCS, UCS2, UCS4, UCS2LE, UCS2BE, UCS4LE, UCS4BE. When no CODEPAGE clause is specified, the database default is assumed (e.g. Single-Byte).

In addition, Transbase® versions newer than 6.2 support more codepage specifications, namely: PROPRIET, SJIS, EUC, and single-byte codepages denoted by a string (enclosed in single quotes) that is a valid server codepage setting, e.g. 'german' or 'greek' on Windows and UNIX platforms. On UNIX platforms, a wide variety of strings is supported including e.g.: 'de_DE', 'de_DE@euro', 'de_DE.utf8' or even 'de_LU.iso885915@euro'. The latter denotes a codepage for Luxemburg in german language with ISO-8859-15 codepage (supporting the Euro sign). The UNIX command `locale -a` usually lists all valid strings.

Furthermore, Transbase® versions newer than 6.2 also provide a `tbmode` statements like `tbmode codepage <codepagespec>` which sets the default specification for SPOOL statements for this session. It can be set before a sequence of (unchanged) SPOOL statements:

```
tbmode codepage <codepagespec> ;
spool <table> from <file> ;
spool <table> from <file> ;
…
```

Finally, the tools tbarc and tbtar have been extended to accept parameters that describe the input interpretation if the archives have been generated from older Transbase® versions. See the release notes for versions 5.3 and 6.2 resp.

2. Converting into UTF-8 databases

2.1. Using tbtar (after-6.2 versions)

This means, the extended codepage specifications of SPOOL statements and Transbase® tools can be used.

Assuming that the original database was of codepage "propriet" just follow these steps:

- Create an archive of your existing database using tbtar:
  - `tbtar -w <db> f=<tbtar_filename> …`
- Create a new database appropriately; specify "utf8" as codepage and specify an appropriate "Locale Setting", e.g. german, either interactively or by the command line options cp=utf8 and loc=german.
- Load the database by tbtar, e.g.:
  - `tbtar -r <db> f=<tbtar_filename> …`
    `locale=propriet`
  - `tbtar -r <db> f=<tbtar_filename> …`
    `locale=greek`
  - `tbtar -r <db> f=<tbtar_filename> …`
    `locale=euc`

`tbtar` uses a tbmode statement and sets the default codepage for SPOOL statements to the value specified by the locale parameter. Thereby all input data is interpreted to be in locale and is converted first into UNICODE (by means of the mbstowcs system call) and then into UTF-8.

Please note that the specified locale= clause must be valid at database server side. Otherwise an error is reported. Under UNIX, the command `locale -a` lists all valid values.

Also note that locale=propriet maps characters one to one into UNICODE, i.e. a character 0xC4 is mapped to U+C4; this is the same as if locale=iso-8859-1 would have been specified.

2.2. Using tbarc (after 6.2 versions)

Using `tbarc` some more flexibility is available since the database contents are saved by spooling tables into files and generating a SQL script that rebuilds the database from the spooled files. Between saving the database and rebuilding it, the script could be modified manually, e.g. in order to adapt some data types. In particular, the spool file contents could be converted manually into the designated new codepage before the database is being rebuilt.

A specific option `cp=` is available both on saving the database and on restoring the database. In the further case, spool files are generated in the specified codepage (which should be the designated codepage). In the latter case, spool files are expected to be of the specified codepage and eventually converted into the destination codepage. Unless this option is specified, the spool files are generated in the codepage of the source database and are expected to be in the codepage of the destination database.

In either case, when a `cp=` option is specified, a `tbmode` codepage statement is issued before spooling any data. Upon `-w` option, the `tbmode` statement can be found at the beginning of `make.db`, upon `-r` option, the `tbmode` statement is issued before `make.db` is executed.

- Create the archive (in a new directory):

```
tbarc -w <dir> <db> [ cp=german]
```

- Eventually modify spool files or database script (`make.db`)
- Rebuild the database from the archive:

```
tbarc -r <dir> <db> [ cp=german]
```

2.3. Using tbtar (pre-6.2 versions)

If your database version is even older than 6.2 you should take one intermediate step:

- Apply `tbtar -w`
- Create a new database in codepage "propriet" with the new Transbase® version (at least of version 6.2)
- Apply `tbtar -r`
- Follow the steps listed under 2.1.

2.4. Using tbarc (pre-6.2 versions)

An analogous procedure is necessary when you are planning to use tbarc; take the following intermediate step:

- Apply `tbarc -w`
- Create a new database in codepage "propriet" with the new Transbase® version (at least of version 6.2)
- Apply `tbarc -r`
- Follow the steps listed under 2.2.

3. Problems

3.1. System Tables

System tables hold information about the database, such as field names, field types, but also view definitions and integrity constraints.

Since identifiers may be arbitrary UTF8 strings, all conversions described for normal tables also refer to system tables. In particular, view definitions also are converted from the input codepage into UTF8.

Special attention must be paid to externally prepared modules (in table sysexternal) which may be Java class files or Java source files or DLLs. These are stored as BLOBs and are not converted at all. Obviously, for class files as well as DLLs this is correct. For source files, however, it depends on their contents and must be performed manually, as in the case of fulltext data stored in BLOBs.

3.2. SQL Script

The SQL script generated by tbarc contains SQL statements necessary to rebuild the database. In particular, it contains table, view or domain definitions as well as integrity constraints etc. All these SQL statements may contain strings or (double quoted) identifiers which may contain non-ASCII characters.

Therefore, the script itself must be converted into UTF-8 before it can be executed in the re-built database. For several reasons, the conversion of this file `make.db` must be done *manually* as noted above under 2.2. Note that this is necessary only when the database contains identifiers or default values or constraints whose codepages extend standard ASCII.

Depending on your operating system, the following tools can be used to convert a file into UTF-8: Under Windows, the notepad editor is able to convert files into UTF-8 using the "SaveAs" button and selecting "UTF-8" as coding; under Linux, the command `iconv [ -f input_coding] -t utf8` can be used to convert the standard input (assumed in the specified coding or in current coding) into UTF-8 standard output.

3.3. BLOBs

BLOBs (binary large objects) are defined to contain binary non-textual data. Therefore BLOBs are never interpreted by Transbase® and do not be converted by any of the above mentioned procedures. Unfortunately, BLOBs can be used as source for fulltext indexes. In that case, the contents of BLOBs are assumed to be textual data and handeled like character strings. Those BLOBs obviously should be converted like other textual data, too. Transbase® cannot know whether a given BLOBs contains textual or binary data and leaves BLOBs always as they are. In other words, BLOBs that are used to hold textual data must be converted manually, as described under 3.2 above. In particular, this must happen before fulltext indexes are created on these BLOBs.

Contact

Transaction Software GmbH
Willy-Brandt-Allee 2
81829 Muenchen, Germany

Tel.: +49 89 / 627 09 - 0
Fax: +49 89 / 627 09 - 11

info@transaction.de
www.transaction.de
www.transbase.de