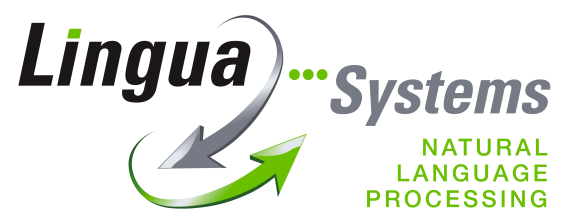


Software Specification

for

AutoUniConv

Covers version 1.1.0



1 Introduction

This document provides the software specification for **AutoUniConv**, version 1.1.0.

AutoUniConv is a C/C++ library that automatically detects and converts text from various character encodings to Unicode.

The specification serves as a general introduction to the software on the one hand and as a detailed description on the library's scope of service on the other hand.

Hint on conventions used:

At several points it is necessary to make a distinction between strings that may not contain embedded NUL characters and those which may, due to their special character encoding, potentially contain NUL characters. The former are called "Strings" while the latter are called "Byte Strings".

2 Overview on the Software

AutoUniConv automatically detects the character encoding of textual input and converts it to a Unicode Transformation Format (UTF-8, UTF-16 and UTF-32).

2.1 User's Requirements

Every user with basic knowledge of C/C++ programming and library usage is able to use **AutoUniConv** right away.

AutoUniConv's field of application is generally spoken every field in which textual data with an unknown character encoding is processed.

2.2 Operating Environment

AutoUniConv's standard version is currently available for the following operating systems:

- Debian GNU/Linux (x86/x86_64): *Lenny* (5.0), *Squeeze* (6.0)
- Ubuntu GNU/Linux (x86/x86_64): *LTS (10.04)*
- Red Hat/CentOS GNU/Linux (x86/x86_64): *RHEL 5*
- FreeBSD (x86/x86_64): *7, 8*
- Microsoft Windows (x86): *XP, Server 2003, Vista, Server 2008, 7*
- Microsoft Windows (x86_64): *7, Server 2008 R2*

The software may very well work on other versions and/or distributions without modification although **AutoUniConv** is only supported in the environments specified above.

Versions for other distributions and/or operating systems may be made available upon request as well.

2.3 Dependencies

AutoUniConv does not require any additional software (libraries) to be installed except for the standard C and thread library shipped with the particular system.

2.4 Use of Resources

The necessary amount of memory (RAM) operatively depends on the size and type of the actual input.

3 Scope of Service

The library automatically identifies the character encoding a text is encoded in and converts it to Unicode. The user may choose from a set of UTF schemes. **AutoUniConv** does not recode the input but encodes a separate output. Additionally the length and the chosen Unicode encoding are returned.

3.1 User Interface

Functions are provided to access the library's functionality. There are two main functions for the conversion of the given input.

Additionally a function for error handling and freeing memory is provided.

All the functions with their parameters and return values are explained in detail in the User Manual to this software.

3.2 Input

3.2.1 Input Format

Only input in plain text format can be processed. The input has to be passed to **AutoUniConv** as either a `String` or a `Byte String`.

One of the functions expects a byte string and needs an additional parameter that specifies the length of the string.

The second function takes a string argument. In this case, the length has not to be given by the user. This function can only be used for 8-bit encodings when there is no NUL-character encoded within the input.

The input's length is only limited by the respective data types.

3.2.2 Supported Character Encodings (Input)

The supported character encodings cover commonly used encodings as well as some legacy ones:

Type	Character Encoding
ISO-8859	ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-8859-5, ISO-8859-7, ISO-8859-15, ISO-8859-16
Windows	Windows-1250, Windows-1251, Windows-1252, Windows-1253, Windows-1257
Macintosh	MacCentralEuropean, MacCyrillic, MacGreek, MacRoman, MacRomanian, MacUkrainian
IBM/DOS Code Pages	CP737, CP775, CP850, CP852, CP855, CP866
Unicode	UTF-8, UTF-16LE, UTF-16BE, UTF-32LE, UTF-32BE
National	Big5, GB2312, KOI8-R, KOI8-U, ASCII

3.3 Processing

The input's character encoding is automatically identified internally and converted to a UTF scheme that can be chosen freely (see chapter 3.4.2).

A minimum input of about 25 characters is recommended to increase accuracy in identifying the character encoding. However, the character encoding may be identified even with less input, but the risk of errors is increased.

If the input is already encoded in a Unicode encoding (UTF-16 or UTF-32) and marked with a Byte Order Mark (BOM), the BOM will be skipped during conversion.

As endianness (Little-Endian or Big-Endian) of UTF-16 and UTF-32 is explicitly chosen when calling one of [AutoUniConv's](#) functions, no BOM will be set.

By using flags, the mode of conversion can be influenced on each function call:

```
AUC_DEFAULT - Default mode
AUC_STRICT  - Throw an error in case of decoding failures
AUC_WARN    - Print a warning in case of decoding failures
```

In default mode, decoding errors are either replaced with a predefined character - the tilde ("~") - or an error is thrown.

3.4 Return Value

The return value is a data structure containing the output encoded in the chosen UTF encoding. The input itself remains unchanged. Additionally the output's length and the specified UTF encoding are provided.

3.4.1 Format

The data structure is formally defined as follows:

```
typedef struct {
    char    * bytes;    /* UTF-Bytes */
    size_t  len;       /* Length */
    auc_utf_t utf;     /* encoded UTF */
} auc_bytes_t;
```

3.4.2 Supported Character Encodings (Output)

The conversion may be done to the following UTF schemes:

Type	Character Encoding
Unicode	UTF-8, UTF-16LE, UTF-16BE, UTF-32LE, UTF-32BE

3.5 Error Handling

[AutoUniConv](#) provides thread-safe error handling facilities. Each thread has its own error indicator that stores numeric error codes. Given these error codes, a natural language error message may be generated by a library function. For convenience, a named constant is defined for each error code, so that error codes can be referenced by name.

3.6 Security Considerations

[AutoUniConv](#) was designed and implemented with security in mind and has no single known safety defect. Besides that, the application does not create any temporary files or evaluate any environment variables.

Memory allocated internally by [AutoUniConv](#) is always freed. This includes all known error paths.

3.7 Thread Safety

[AutoUniConv](#) is thread-safe: more than one thread may call the library's functions at the same time. Each thread takes care of its own error handling and allocates two variables on the *Thread-Local Storage* (TLS).

4 Restrictions

- At the state of the art, an identification of the character encoding cannot be guaranteed to be accurate in any case without restrictions. There may be documents that lead to wrong identifications - resulting in failures in the conversion as well.
- An input can only be processed accurately if it is available in plain text or has been preprocessed to this format before. Any such preprocessing is not part of [AutoUniConv](#).
- Only supported character encodings can be processed. Unsupported character encodings will be processed as well but are likely to result in encoding failures.
- The software itself is provided in English only.
- The manual is only provided as a PDF document, in both an English and German version.
- There is no general guarantee for downward compatibility to previous versions.

5 Deliveries

A complete installation of the software consists of the compiled library, all associated header files, man pages and the user manual in an English and German version. The installation requires about 3 MiB of disk space.

6 References

- User Manual for [AutoUniConv](#) version 1.1.0
- RFC 2781: UTF-16, an encoding of ISO 10646
- RFC 2279: UTF-8, a transformation format of ISO 10646
- The Unicode 5.0 Standard