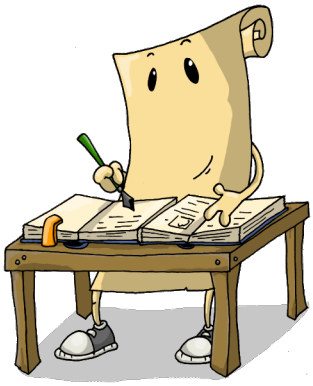
User Manual

for

Alphawandler SDK

A software for transliteration

Covers version 1.0.0



Lingua Systems
NATURAL LANGUAGE PROCESSING

Alphawandler User Manual, published April 23, 2014.

Lingua-Systems Software GmbH, Gerichtsstraße 42, 44649 Herne, Germany, *info@lingua-systems.com*

Contents

About this Manual

This manual addresses users with experience in C/C++ programming and at least a basic knowledge of library usage.

The manual provides a short introduction to the library, followed by instructions how to install the Alphawandler software package. Afterwards some hints on the usage of a transliteration system are given, before the complete interface (API) is introduced along with the possibilities of error handling.

For a quick start have a look at the documentation of the application programming interface (chapter 4 on page 8).

Administrators who want to install the software can obtain all necessary information from chapter 2, page 5.

1. Introduction

Alphawandler is a software for transliteration that transfers text from one writing system or alphabet to another one.

Every transliteration is - wherever possible - done according to a given transliteration standard as defined by various national or international organisations, like ISO, DIN or GOST. Besides that, common national transliteration rules are included (see chapter 3 for details).

If the chosen standard allows to, the transliteration can be used bidirectional.

An intuitive interface to the library allows you to integrate Alphawandler easily. The C/C++ library is thread-safe and provides access to all functions needed to make use of a transliteration within your own application.

Any input passed to Alphawandler has to be plain text and should be encoded in UTF-8.

2. Installation

2.1. Requirements

Alphawandler merely requires the system's standard C runtime environment.

2.2. What will be installed

The Alphawandler SDK contains a dynamic library (DLL/SO), its header file, the code of an example application and this manual.

The Software Development Kit for Linux contains the following files:

```
./doc:
example.c   LICENSE.txt   manual-sdk-eng.pdf

./include:
aw.h

./lib:
libaw.so   libaw.so.1   libaw.so.1.0.0
```

2.3. Installing the Software

Alphawandler SDK is provided as a compressed archive, either in "Zip" or "tar.gz" form, depending on the target platform.

To install the software, just unpack the archive to a directory of your choice and add the library and header files to your project.

2.4. Deinstalling the Software

To deinstall the software, just remove the directory you unpacked Alphawandler SDK to.

3. Hints on the Usage of Alphawandler SDK

Only input in plain text format can be processed, that is encoded in UTF-8.

The transliteration is based on standards. If the input contains characters that do not belong to the writing system covered by the standard, they will remain unchanged in the output.

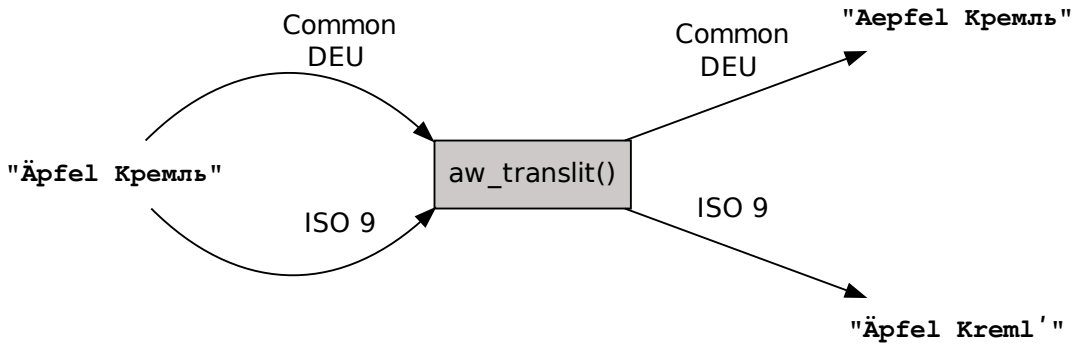


Figure 1: Minimal Transliteration Example

Some transliterations can only be applied in one direction. These lossy transliterations are not reversible, because the mapping of characters is ambiguous.

Example: The German word *Äpfel* would be transliterated as *Aepfel* according to the "'Common DEU"' standard - but it is not possible to reverse the direction of the transliteration, because the character sequence *ae* is also common without denoting the umlaut *ä*, as in *Michael* or *Tetraeder*.

3.1. Supported Writing Systems and Standards

Cyrillic

Standard	Description	Reversible
ALA-LC RUS	Cyrillic to Latin, Russian	no
ISO 9	Cyrillic to Latin	yes
DIN 1460 RUS	Cyrillic to Latin, Russian	yes
DIN 1460 UKR	Cyrillic to Latin, Ukrainian	yes
DIN 1460 BUL	Cyrillic to Latin, Bulgarian	yes
Streamlined System BUL	Cyrillic to Latin, Bulgarian	no
GOST 7.79 RUS	Cyrillic to Latin, Russian	yes
GOST 7.79 RUS OLD	Cyrillic to Latin with support for Old Russian (pre 1918), Russian	no
GOST 7.79 UKR	Cyrillic to Latin, Ukrainian	no

Greek

Standard	Description	Reversible
ISO 843	Greek to Latin	no
DIN 31634	Greek to Latin (academic)	no
Greeklish	Greek to Latin (phonetic)	no

Latin

Standard	Description	Reversible
Common CES	Czech without diacritics	no
Common DEU	German without umlauts/sz-ligature	no
Common POL	Unaccented Polish	no
Common RON	Romanian without diacritics	no
Common SLK	Slovak without diacritics	no
Common SLV	Slovenian without diacritics	no

Additional languages can be added upon request.

4. Application Programming Interface

4.1. Overview

The C/C++ library contained in the Alphawandler SDK provides an API that is intuitive to use and allows integration into applications easily. All functions and data structures are prefixed *aw_* to avoid confusions and collisions with other third party library functions and are defined in the header file *aw.h*.

Input passed to the library is expected to be plain text and encoded in UTF-8.

aw_strerror() provides an English error messages for each error code.

aw_version() and *aw_version_string()* provide the library's version at runtime.

All functions are thread-safe and may be called from multiple threads simultaneously.

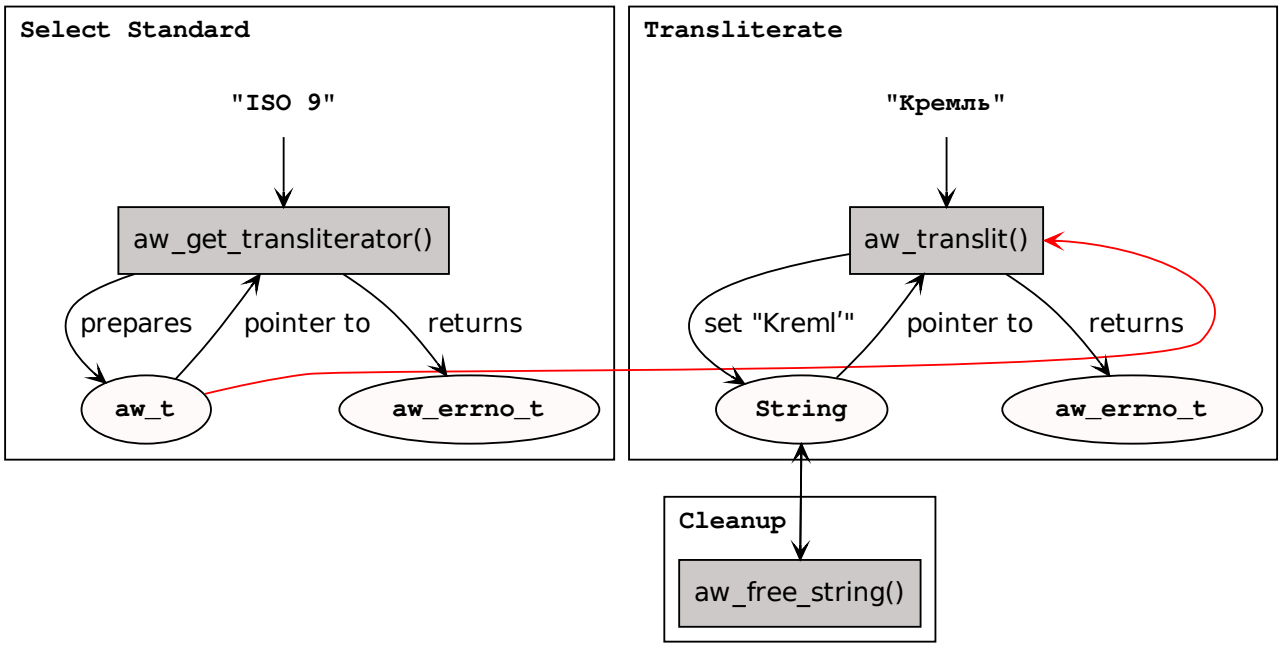


Figure 2: Transliteration

4.2. Important Data Structures

The data structure `aw_errno_t` is described in a separate chapter on error handling (chapter 4.4, page 12).

4.2.1. Alphawandler-Object aw_t

The data structure `aw_t` contains data that is exclusively used by Alphawandler internally. No application should evaluate or change the data directly.

An `aw_t` object represents a transliterator that is required by *aw_translit* in order to transliterate a given input string.

First, you should assign the macro `AW_T_INITIALIZER` to any variable of type `aw_t` on declaration in order to initialize it with its default values.

The functions *aw_get_transliterator()* and *aw_get_transliterator_by_number()* then prepare an `aw_t` object to be used to transliterate according to a given standard. An `aw_t` object is expected as an argument by almost every Alphawandler main function.

4.2.2. Transliteration Options aw_opt_t

The following options are provided and can be passed to *aw_translit()*:

Option	Meaning
`AW_DEFAULT`	Use defaults
`AW_REVERSE`	Reverse transliteration direction
`AW_NO_VALIDATE`	Disable input validation

Figure 3: `aw_opt_t` Options and their Meaning

If `AW_DEFAULT` is given, transliteration is done in the forward direction (that is to Latin) with enabled UTF-8 validation.

If the option `AW_REVERSE` is passed to *aw_translit()* although the chosen transliteration standard does not support reverse transliteration, `AW_ENOREV` will be returned as an error indicator.

`AW_NO_VALIDATE` disables UTF-8 input validation and may have a positive impact on execution speed. However, this option should only be considered if the input is already known to consist of valid UTF-8 octets only.

4.2.3. Transliteration Standard Information aw_info_t

Information on a transliterator and the underlying transliteration standard can be retrieved after selecting a standard with either *aw_get_transliterator()* or *aw_get_transliterator_by_number()*.

Each variable of type `aw_info_t` should be initialized using the `AW_INFO_T_INITIALIZER` macro right on its declaration.

The data structure provides some details on the selected `aw_t` transliterator:

1. Name of the transliteration standard
2. Short description of the standard
3. Reversibility indicator

The formal definition of the data structure is as follows:

```
typedef struct aw_info
{
    const char *name;
    const char *description;
    enum { AW_NON_REVERSIBLE, AW_REVERSIBLE } reversible;
} aw_info_t;
```

If `reversible` is set to `AW_REVERSIBLE`, the transliteration direction can be reversed, `AW_NON_REVERSIBLE` indicates that this is not possible.

4.3. Function Reference

All of Alphawandler's functions and data structures are defined in the header file *aw.h*. The header has to be included in all applications that make use of the following functions.

Code for an example application covering Alphawandler's main functions is included in this manual (see appendix A on page 14) and in the software distribution.

4.3.1. aw_translit()

```
aw_errno_t  aw_translit(
                const  aw_t    aw,
                const  char   *input,
                aw_opt_t       options,
                char         **output
            );
```

aw_translit() transliterates an input string *input* according to the transliteration standard set for *aw* using the passed *options* and stores the address of the transliterated output string to the address *output* points to.

The first argument (*aw*) has to be an initialized Alphawandler object of `aw_t` type (see chapter 4.2.1 on page 8). The object must have previously been assigned a transliteration standard using either *aw_get_transliterator()* or *aw_get_transliterator_by_number()*.

The *input* has to be UTF-8 encoded and properly terminated.

The mode of operation may be set using the *options*: `AW_DEFAULT` or `AW_REVERSE` (see 4.2.2 on page 9).

The functions return an error code that indicates whether the respective function succeeded (`AW_OK`) or an error occurred. For details on error handling see chapter 4.4 on page 12.

The function allocates memory for the transliterated string and assigns its address to the object pointed to by *output*. The used memory may be freed using *aw_free_string()*, which is mandatory on Windows systems.

The functions is thread-safe and can thus be used by more than one thread at a time.

4.3.2. aw_get_transliterator() and aw_get_transliterator_by_number()

```
aw_errno_t  tr_get_transliterator(
                const  char   *name,
                aw_t         **aw
            );

aw_errno_t  tr_get_transliterator_by_number(
                size_t    number,
                tr_t    **aw
            );
```

Both functions assign a transliterator for a transliteration standard to an Alphawandler object. The standard may either be referenced by its *name* or its *number*.

Named references of standards (see chapter 3 on page 6) are case-insensitive. Besides that, underscores and hyphens may be used instead of blanks (e.g. iso_9 or iso-9).

The functions return an error code that indicates whether the respective function succeeded or an error occurred. For details on error handling see chapter 4.4 on page 12.

The function *aw_get_transliterator_by_number()* is mostly useful to iterate over all transliterators provided for the set of supported standards. When iterating, start with one (1) and continue until the function returns `AW_ESTD` instead of `AW_OK`.

Both functions are thread-safe and thus can be used by more than one thread at a time.

The internal numbering of transliterators may change from release to release. Therefore, only *aw_get_transliterator()* should be used if an individual transliterator should be selected.

4.3.3. aw_get_info()

```
aw_errno_t  aw_get_info(
                    const aw_t aw,
                    aw_info_t *info
            );
```

The function stores information on an Alphawandler object *aw* into an `aw_info_t` variable pointed to by *info*.

The first argument (*aw*) has to be an initialized Alphawandler object of `aw_t` type (see chapter 4.2.1 on page 8). The object must have previously been assigned a transliteration standard using either *aw_get_transliterator()* or *aw_get_transliterator_by_number()*.

The function returns an error code that indicates whether the respective function succeeded (`AW_OK`) or an error occurred. For details on error handling see chapter 4.4 on page 12.

The function is thread-safe and can thus be used by more than one thread at a time.

The variable of type `aw_info_t` that is pointed to by *info* should be initialized using `AW_INFO_T_INITIALIZER` on its declaration.

The function does not need to allocate any memory for the `aw_info_t` data structure or its members. As a result, code using this function does not need to free memory of the structure or its members either.

4.3.4. aw_version() and aw_version_string()

```
unsigned int aw_version(void);

const char * aw_version_string(void);
```

The functions do not take an argument and return the version of the Alphawandler SDK in a numeric or character-based representation.

The functions are thread-safe and thus can be used by more than one thread at a time.

The memory of the string returned by *aw_version_string()* must not be freed.

4.3.5. aw_strerror()

```
const char * aw_strerror(
                aw_errno_t errnum
             );
```

The function takes an error indicator *errnum* of type `aw_errno_t` as an argument and returns a pointer to a read-only string (`const char *`) containing the English error message.

A list of all error codes and descriptions is given in chapter 4.4.1 on page 13.

The function is thread-safe and thus can be used by more than one thread at a time.

The memory of the returned string must not be freed.

4.4. Error Handling

Alphawandler provides an easy to use way to handle errors by evaluating the return value. Every function that may fail has an error indicator as a return value.

Any application that uses Alphawandler should evaluate this error indicator to implement an adequate error handling. The return value `AW_OK` indicates that the function was successful.

Error messages may be obtained using *aw_strerror()* (see chapter 4.3.5 on page 12).

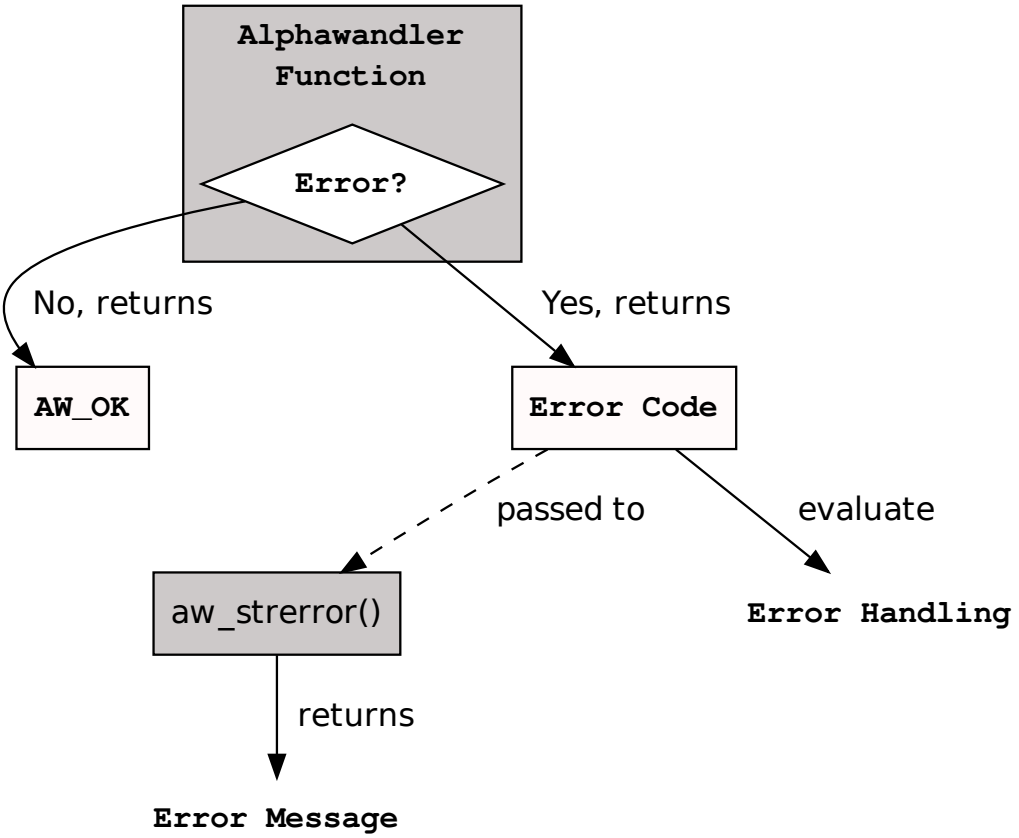


Figure 4: Flowchart of Alphawandler's Error Handling

4.4.1. aw_errno_t Named Error Constants

Alphawandler uses the data structure `aw_errno_t` to provide named error constants for all error cases.

The following table comprises all named error constants used in Alphawandler SDK, accompanied by the error messages returned if passed to *aw_strerror()*.

Constant	Error Message
AW_OK	No error
AW_ENOMEM	Failed to allocate memory
AW_EARG	Invalid argument
AW_EUINV	Invalid unicode
AW_ESTD	No such transliteration standard
AW_ENOREV	Standard is not reversible

Figure 5: `aw_errno_t` Named Constants and Error Messages

4.5. Hints on Application Development

4.5.1. Determining Alphawandler SDK's Version

After including the *aw.h* header, the following preprocessor definitions are available *at compile time*.

Definition	Value
AW_VERSION_MAJOR	1
AW_VERSION_MINOR	0
AW_VERSION_PATCH	0

Figure 6: Version Information at Compile Time

To determine Alphawandler's version at runtime, use *aw_version()* or *aw_version_string()*.

A. Example Application: example.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <aw.h>


static
aw_errno_t translit(const char *std, const char *txt, char **out)
{
    aw_t        aw  = AW_T_INITIALIZER;
    aw_errno_t err = AW_OK;

    if ((err = aw_get_transliterator(std, &aw)) != AW_OK)
    {
        fprintf(stderr, "Error: %s (%d)\n", aw_strerror(err), err);
        return err;
    }

    if ((err = aw_translit(aw, txt, AW_DEFAULT, out)) != AW_OK)
    {
        fprintf(stderr, "Error: %s (%d)\n", aw_strerror(err), err);
        return err;
    }

    return AW_OK;
}


int main(int argc, char *argv[])
{
    char *out = NULL;

    if (argc != 3)
    {
        fprintf(stderr, "usage: %s standard text\n", argv[0]);
        return EXIT_FAILURE;
    }

    if (translit(argv[1], argv[2], &out) == AW_OK)
    {
        puts(out);
        aw_free_string(out);
        return EXIT_SUCCESS;
    }

    return EXIT_FAILURE;
}
```

B. References

→ Lingua-Systems' Alphawandler website,
http://www.lingua-systems.com/transliteration/

→ Alphawandler software specification for version 1.0.0

→ The Unicode Standard,
http://www.unicode.org/

→ RFC 2279: "UTF-8, a transformation format of ISO 10646",
http://www.ietf.org/rfc/rfc2279.txt

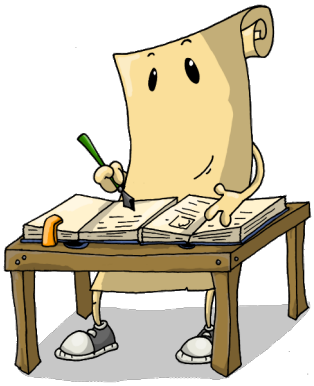
→ ALA-LC Romanization Tables,
http://www.loc.gov/catdir/cpso/roman.html

→ International Organization for Standardization,
http://www.iso.org/

→ German Institute for Standardization,
http://www.din.de/

→ Federal Agency on Technical Regulating and Metrology (GOST),
http://www.gost.ru/

→ Bulgarian Academy of Sciences,
http://www.bas.bg/



http://www.lingua-systems.com/transliteration/

Index