Transtalo Translation System

Edition 0.1, for Transtalo 0.3
Last updated July 2005

by Tijmen Baarda

This file documents the Transtalo Translation System with its library and user command, for automatic translation.

Table of Contents

1 What Is Transtalo?

The Transtalo Translation System, or Transtalo for short, is a free automatic translator written in C++ and partly in C. It allows you to automatically translate documents, web sites, books... And the computer does it for you.

I know very well that computer translating can never be perfect. Translation is work of man. And it will stay so. But imagine. You get a text written in Italian. You know some words of that language, but you can't understand the text. If you translate it into your own language (or a supported language you can read), you will be able to understand the text, although the translation is not perfect.

It is also possible you want to send a letter to a friend, but you don't speak his language and vice versa. So you write your letter in your own language (or, if Transtalo doesn't provide support for it, another language you know), let Transtalo translate it and send it to your friend. He will be able to read your letter, although it again is not perfect.

Transtalo isn't meant as a perfect alternative to human translation. Human translation will always stay the best. But Transtalo will do its best to generate a translation as well as possible.

Reading this book, you should realize that not all features mentioned are implemented yet. For now, see it as a preview to the system.

1.1 The Translation Progress

The translation progress that Transtalo uses is somewhat different from the way used by Babelfish and others. If you aren't interested in the way texts are translated, you may want to skip this paragraph.

1. The Transtalo library can only translate one sentence at once, so the text is separated into sentences.
2. Each sentence is sent to the Transtalo library.
3. The library calls an input module (an external program) to parse the sentence from the source language. It translates single words into Esperanto (this is an artifical language that is completely regular), and puts everything in an XML file. The XML file describes the whole parsed sentence in a language-independent way.
4. After this, the Transtalo library calls the output module that is specialized in the chosen target language, to translate the XML sentence file into a real sentence in the target language.
5. The library receives the sentence as it was translated by the output module.
6. This is done for all sentences in the text.

1.2 Highlights of Transtalo

The Transtalo Translation System has different highlights. They are explained in this section.

1.2.1 Separated Input and Output Modules

Because the input and output parts are independently of each other implemented, we don't need to make modules for *combinations* of source and target languages, but only for the input and output parts for each language.

Maybe you find this somewhat complicated. I'll give an example. You want to translate a text from English to twenty other languages. If each combination of languages needs an own module, a module would be needed for English to French, English to Spanish, English to German, English to Japanese, English to Danish, and so on.

This wouldn't be a problem if we *only* had to translate from English. Then twenty modules would be needed. But imagine somebody from Portugal wants to do exactly the same, except that the source language is Portugese, not English. Then twenty new modules would be needed: from Portugese into English, Portugese into German...

Conclusion: this way it is impossible to have support for all languages. For example: for 70 languages to be fully supported, we would need 70 / 70 * 2 = 2450 (TODO: I don't think that this is correct) modules (we devide it by two because we don't need to have modules that translate into itselves (e.g. English into English, Danish into Danish)).

You could say: "But if can do everything via the English language, you don't need to have so much modules, do you?" That is true, but you know that automatic translation is never perfect. By the first translation (to English), the quality of the text makes a great step backwards. During the second translation, the same thing happens. Also, many information may gets lost.

Transtalo uses a very different approach. It haves different modules for the translation from and translation to parts, called *input* and *output* modules. The input module translates a sentence from the source language, and the output module translates it into the target language.

This way, we only need to have 140 modules to fully support 70 languages: 70 input modules and 70 output modules.

1.2.2 Dialects

Transtalo has support for dialects. It depends on the specific module if it supports the desired dialect.

At this time, there is no single module that supports dialects.

1.2.3 Diagnostic Messages

It is possible that you see warnings and/or errors while translating. If a module sees something strange in the sentence it translates, it will output a message. If it leads to a bad translation, you are encouraged to report it.

Beside warnings and errors it is also possible to switch on information messages on. These give information on the translation progress, such as which phrase is currently translated. These messages are switched off by default, because in most cases you won't like to see them.

1.2.4 Idiom

The translation modules are able to support translating idiom in a good way. In general this work is done by the output module.

It works in the following way. The output module checks the already parsed sentence on all known idiom that is specific for the original language. If it finds something, the output module will try to transform it so it turns in a normal sentence. After that, the sentence is translated normally with the difference that the idiom is (should be) translated well.

1.2.5 Library Interface

The Transtalo Translation System is not only a standalone program (the `transtalo` command), but comes also with a library interface. With this interface, you are able to make applications that use the library for its purposes.

You could e.g. make a graphical user interface for Transtalo, or a web interface. You are not limited to the simple `transtalo` command.

2 Invoking the `transtalo` Command

The `transtalo` command provides a simple user interface to the Transtalo library. It currently only has support for single sentences, whole text translation support will be added later.

`transtalo` uses commands to specify what it must do.

Normal Translation

The command for translating a sentence from one language to another, is `translate` or `l2l` (language to language):

```
transtalo translate|l2l source-lang dest-lang sentence
```

source_lang is the source language, *dest_lang* the target language, and *sentence* is the sentence to be translated. For example

```
transtalo translate en de "This is a sentence."
```

should output "Dies ist ein Satz." (but it doesn't at this time as there neither is an English input module nor a German output module).

Human Language to XML File

If you want to translate only *from* the source language and want to stop after that, outputting the XML file, use the `lang2xml` or `l2x` commands:

```
transtalo {lang2xml|l2x} source-lang sentence [output-file]
```

Where *output-file* is the output location for the XML file.

XML File to Human Language

If you only want to do the translation *to* the target language using an XML sentence file, use the `xml2lang` or `lang2xml` commands:

```
transtalo {xml2lang|x2l} xml-file dest-lang
```

This will output the final sentence.

2.1 Options

The following table lists the options for the `transtalo` command.

'`--help`'
'`-h`' Displays information about the command and exits

'`--info-inputmodule` *`language`*'
Displays information about the input module for *language*

'`--info-outputmodule` *`language`*'
Displays information about the output module for *language*

'`--dialect-source` *`dialect`*'
Use *dialect* as dialect for the source language

'`--dialect-dest` *`dialect`*'
Use *dialect* as dialect for the target language

‘`--list`’
‘`-l`’ Lists all installed modules

‘`--quiet`’
‘`-q`’ Display neither messages nor errors while translating

‘`--verbose`’
‘`-v`’ Display much information about the translation progress

‘`--version`’
‘`-V`’ Displays the version information about the command and exits

2.2 Language Codes

The language codes used come from the ISO 639 specification. If you want to know which code you have to use, you can type ‘`transtalo --list`’ to see which languages are installed: you’ll see both the codes and full names.

3 Using the User Library

The user library is an easy to use C interface to the Transtalo Translation System. It actually is a convience library, since it is possible to call the translation modules by hand.

The transtalo user library consists of a number of functions, enumerations and defines. Besides the translation, it also allows looking up information about the available translation modules.

The library's name is `libtranstalo`, you can use it by adding `-ltranstalo` to the command line while linking.

3.1 The Functions

3.1.1 `transtalo_translate`

```
#include <transtalo.h>

int transtalo_translate(char *source_lang,
  char *source_lang_dialect, char *dest_lang, char *dest_lang_dialect,
  char *sentence, char *target, int verbosity);
```

This function is the library equivalent of the `translate` command from the `transtalo` command line interface.

It translates *sentence* from *source_lang* with *source_lang_dialect* as dialect into *dest_lang* with *dest_lang_dialect* as dialect and puts it into *target*.

If *source_lang_dialect* and/or *dest_lang_dialect* are NULL, they will be ignored. *target* must be an initialized string (1000 characters—this will change in the future).

3.1.2 `transtalo_lang2xml`

```
#include <transtalo.h>

int transtalo_lang2xml(char *source_lang,
  char *source_lang_dialect, char *sentence, char *xml_file,
  int verbosity);
```

This function is the library equivalent of the `lang2xml` command from the `transtalo` command line interface.

It translates *sentence* from *source_lang* with *source_lang_dialect* as dialect into an XML file and puts it into the *xml_file* file.

If *source_lang_dialect* is NULL, it will be ignored.

3.1.3 `transtalo_xml2lang`

```
#include <transtalo.h>

int transtalo_xml2lang(char *xml_file, char *dest_lang,
  char *dest_lang_dialect, char *target, int verbosity);
```

This function is the library equivalent of the `xml2lang` command from the `transtalo` command line interface.

It translated the *xml_file* XML file into *dest_lang* with *dest_lang_dialect* as dialect and puts it into *target*.

If *dest_lang_dialect* is NULL, it will be ignored. *target* must be an initialized string (1000 characters—this will change in the future).

3.1.4 transtalo_get_modules

```
#include <transtalo.h>

void transtalo_get_modules(char *buffer[], int type);
```

Get all supported modules of *type* and put it into *buffer*. *type* is one out of TS_INPUT_MODULE and TS_OUTPUT_MODULE.

buffer must be an initialized array of pointers, for example:

```
char *modules[100];
```

This will change in the future.

3.1.5 transtalo_get_long_name

```
#include <transtalo.h>

char *transtalo_get_long_name(const char *lang);
```

This function returns the full name for the ISO language code in *lang*.

3.1.6 transtalo_get_info

```
#include <transtalo.h>

TRANSTALO_MODULE_INFO *transtalo_get_info(int module_type,
char *language);
```

This function puts information about the specified translation module into a pointer to a TRANSTALO_MODULE_INFO structure.

module_type is one out of TS_INPUT_MODULE and TS_OUTPUT_MODULE, and specifies wether you want respectively an input or an output module.

language is a string containing the language code you want to get the information from.

The function returns a struct TRANSTALO_MODULE_INFO declared in <transtalo.h> as

```
typedef struct {
  char *short_language;
  char *long_language;
  char *version;
  int state;
  char *authors[16];
  char *contributors_name[16];
  int *contributors_function[16];
} TRANSTALO_MODULE_INFO;
```

In the struct *state* defines how well the module can translate, and is a value from 0 up to 5.

contributors_function is an array which defines the functions of the contributors, anolog to the ones in *contributors_name*, and is: TS_FUNCTION_UNKNOWN if the function was not specified, TS_FUNCTION_BUG_REPORT if it was a bug reporter, TS_FUNCTION_FEATURE_REQUEST if he or she requested a function, TS_FUNCTION_PATCH if it was someone who provided a patch.

3.1.7 transtalo_get_release

```
#include <transtalo.h>

char *transtalo_get_release();
```

Returns the release information of the current core library, for example: "Transtalo Translation System, release 0.3.0".

3.1.8 transtalo_get_copyright

```
#include <transtalo.h>

char *transtalo_get_copyright();
```

Returns copyright information of the current core library, for example: "Copyright (C) 2004, 2005 the Transtalo team".

3.1.9 transtalo_get_version

```
#include <transtalo.h>

char *transtalo_get_version();
```

Returns the version of the current core library, for example: "0.3.0".

3.2 Verbosity

The *verbose* variable that is uses in some functions can be one out of:

'TS_QUIET'
: Quiet, no messages are output

'TS_NORMAL'
: Normal, only warnings and errors are output

'TS_VERBOSE'
: Verbose, warnings and errors are output, but information messages too

3.3 Return Values

The first 3 functions that deal with a translation return an integer that is defined as one out of the following:

'TS_NO_ERROR'
: no errors (value 0)

'TS_SOURCELANG_MISSING'
: the source lang is missing

'TS_SOURCELANG_DIALECT_MISSING'
: the source lang dialect is missing

'TS_DESTLANG_MISSING'
: the destination lang is missing

'TS_DESTLANG_DIALECT_MISSING'
: the destination lang dialect is missing

'TS_TRANSLATION_INPUT_FAILURE'
: fatal error in input module

'TS_TRANSLATION_OUTPUT_FAILURE'
: fatal error in output module

'TS_FORK_FAILED'
: process fork failed

'TS_PIPE_FAILED'
: making a pipe failed

4 The SENTENCE Class and Its Children

The class SENTENCE is the central part of each module. The goal of an input module is filling such a class with all information about the sentence so that it can be used by an output module to generate a sentence.

Input modules export the class to an XML sentence file, which is read by output modules to reform the class.

But what's in the class? The full parsed sentence. It contains the subject, object, adverbials, finite forms and so on. This chapter will explain the way the class is built up.

Studying the class you may want to read the appropiate header file as well. It is '`transtalo_objects.h`'. Because it is automatically included by '`transtalo_modules.h`', you don't need to include it if you are making a translation module.

> **Note:** reading this chapter, it is important to know that the classes aren't complete yet and are always about to change.

4.1 Abbreviations

The SENTENCE class and its children use much abbreviations for the phrase's names. These are listed here:

i_object indirect object

d_object direct object

subcomp subject complement

adverbial_adverb
: adverbial consisting of an adverb

adverbial_preposition
: adverbial consisting of a preposition with an object

pers_pronoun
: personal pronoun

ind_pronoun
: indicating pronoun

4.2 SENTENCE

This is the main class that describes the whole sentence. Let's start with the declaration.

```
class SENTENCE
{
  public:
    vector<OBJECT> subjects, i_objects, d_objects;
    vector<SUBCOMP> subcomps;
    PREDICATE predicate;

    vector<ADVERBIAL_ADVERB> adverbials_adverb;
    vector<ADVERBIAL_PREPOSITION> adverbials_preposition;
```

```
        bool negative;
        bool asking;

        string original_language;
        string original_sentence;

        void add_subject(OBJECT subject);
        void add_i_object(OBJECT i_object);
        void add_d_object(OBJECT d_object);
        void add_subcomp(SUBCOMP subcomp);
        void add_finform(FINFORM finform);
        void add_infinitive(INFINITIVE infinitive);
        void add_adverbial_adverb(ADVERBIAL_ADVERB adverbial_adverb);
        void add_adverbial_preposition(ADVERBIAL_PREPOSITION adverbial_preposition);

        int save(string filename);
        void load(string filename);

        SENTENCE();
    };
```

Most members are self-explaining. The following list contains information about members that need more information:

`original_language`
: contains the original language before the translation (ISO language code).

`original_sentence`
: contains the original sentence before the translation.

`negative` is true if the sentence is negative, e.g. in "He isn't able to sell fish."

`asking` is true if the sentence is asking, e.g. in "Do you see him?"

`int save(string filename);`
: saves the SENTENCE to an XML file named *filename*.

`void load(string filename);`
: loads the XML file named *filename*. Throws the `parse_error` exception if an XML parsing error was noticed.

4.3 OBJECT

`OBJECT` is the general class for objects. It can be a subject, indirect object, direct object, an object belonging to a subject complement, or an object in an adverbial after the preposition.

An object can be an object with a noun (e.g. 'the big cat'), this is called a noun object; or it can be a personal pronoun (e.g. 'you'), or it can be an independent indicating pronoun ('this' or 'that').

```
    namespace object_type {
      enum OBJECT_TYPE {
```

```
    undefined,
    noun_object,
    pers_pronoun,
    ind_pronoun
  };
}

class OBJECT {
  public:
    object_type::OBJECT_TYPE type;
    NOUN_OBJECT noun_object;
    PERS_PRONOUN pers_pronoun;
    IND_PRONOUN ind_pronoun;

    RELATION relation;

    OBJECT(object_type::OBJECT_TYPE type = object_type::undefined);
    OBJECT(NOUN_OBJECT noun_object);
    OBJECT(PERS_PRONOUN pers_pronoun);
    OBJECT(IND_PRONOUN ind_pronoun);
};
```

An OBJECT should have only one out of `noun_object`, `pers_pronoun` and `ind_pronoun`. `type` must confirm with the actual type that is used.

The three last constructors automatically add the contents and set the right type belonging to the class types they accept. The first constructor only sets the type.

See the section `RELATION` for information about `relation`.

4.4 NOUN_OBJECT

This class is always child of `OBJECT` and represents an object with a noun and possible adjectivals, adverbials and indicating pronouns.

```
class NOUN_OBJECT
{
  public:
    string noun, noun_orig;
    bool noun_known;

    vector<ADJECTIVAL> adjectivals;
    vector<ADVERBIAL_PREPOSITION> adverbials_preposition;

    vector<IND_PRONOUN> ind_pronouns;

    bool definite;
    bool force_no_article;
    bool plural;
    bool little;
```

```
        int begin_pos, end_pos;

        void add_adjectival(ADJECTIVAL adjectival);
        void add_adverbial_preposition(ADVERBIAL_PREPOSITION adverbial_preposition);
        void add_ind_pronoun(IND_PRONOUN ind_pronoun);
        NOUN_OBJECT();
    };
```

The following members need more information:

`noun` the noun in Esperanto as it was translated by the input module

`noun_orig`
: the original noun before the translation

`noun_known`
: is true if the noun was known by the input module

`definite` is true if the noun is definite (i.e. when it was called before, in English this means the article 'the' is used instead of 'a(n)'.

`force_no_article`
: is true if the output module needs to be used not to use an article

`little` is true if the noun is in its little form—in most languages, including English, this needs to be translated by adding the adjective 'little'. Some languages have own suffixes for them, e.g. '-chen' and '-lein' in German.

`begin_pos`
: the position of the character at the beginning of this phrase in the original sentence

`end_pos` the position of the character at the end of this phrase in the original sentence

4.5 PERS_PRONOUN

This class is always child of `OBJECT` and represents an object consisting of a personal pronoun.

```
    class PERS_PRONOUN {
      public:
        int person;
        bool plural;
        int gender;
        bool definite;
        bool polite;

        vector<ADJECTIVAL> adjectivals;
        vector<ADVERBIAL_PREPOSITION> adverbials_preposition;

        vector<IND_PRONOUN> ind_pronouns;
```

```
        int begin_pos, end_pos;

        void add_adjectival(ADJECTIVAL adjectival);
        void add_adverbial_preposition(ADVERBIAL_PREPOSITION adverbial_preposition);
        void add_ind_pronoun(IND_PRONOUN ind_pronoun);
        PERS_PRONOUN(int person=0, bool plural=false, int gender=0);
    };
```

`person` the person; can be 1, 2 or 3 (1st, 2nd, 3rd person)

`plural` is true if the number is plural

`gender` the gender; can be 0 (neuter), 1 (male), 2 (female)

`definite` well I don't know either what I meant with this

`polite` is true if the personal pronoun should be used polite; most languages only support it for 2nd person (the German 'Sie' instead of 'du'), and much languages, including English, don't even support it at all

See the explanation for `begin_pos` and `end_pos` in the `SUBJECT` section.

4.6 IND_PRONOUN

This class can be child of `OBJECT`, but also of `NOUN_OBJECT` or `PERS_PRONOUN`. It represents an either dependent or independent indicating pronoun ('this', 'that', 'these' in English).

```
    namespace ind_pronoun_type {
      enum IND_PRONOUN_TYPE {
        undefined, t_this, t_that
      };
    }

    class IND_PRONOUN {
      public:
        ind_pronoun_type::IND_PRONOUN_TYPE type;

        int begin_pos, end_pos;
        RELATION relation;

        IND_PRONOUN(ind_pronoun_type::IND_PRONOUN_TYPE type = ind_pronoun_type::undefined)
    };
```

`type` must be one out of `ind_pronoun_type::t_this` and `ind_pronoun_type::t_that`. It should not stay at `ind_pronoun_type::undefined`.

4.7 ADJECTIVAL

This class represents an adjectival.

```
    class ADJECTIVAL {
      public:
        string adjective;
```

```
        string adjective_orig;
        bool adjective_known;

        int degree;

        vector<ADVERBIAL_ADVERB> adverbials_adverb;

        int begin_pos, end_pos;
        RELATION relation;

        void add_adverbial_adverb(ADVERBIAL_ADVERB adverbial_adverb);
        ADJECTIVAL();
    };
```

`adjective` is the adjective in Esperanto, `adjective_orig` the original word. `adjective_known` indicates if it was known by the input module.

`degree` is one out of 0 ('big'), 1 ('bigger') or 2 ('biggest').

4.8 ADVERBIAL_ADVERB

This class represents an adverbial with an adverb. It can only be child of `ADJECTIVAL` or `SENTENCE`. An example is 'really' in 'It is a really big file' or 'well' in 'She shings well'.

```
    class ADVERBIAL_ADVERB {
      public:
        ADJECTIVAL adverb;

        int begin_pos, end_pos;
        RELATION relation;

        ADVERBIAL_ADVERB::ADVERBIAL_ADVERB();
    };
```

As you can see the adverb is formed by an `ADJECTIVAL` object. This may sound weird, but the adjectival has the same properties as the adverb, so this is possible.

4.9 ADVERBIAL_PREPOSITION

This class represents an adverbial with a preposition and an object. It can be child of an `OBJECT` or `SENTENCE`. An example is 'in the room' in 'The children are in the room'.

```
    namespace place_in_sentence {
      enum PLACE_IN_SENTENCE {
        undefined,
        start_of_sentence,
        after_subject,
        after_d_object,
        after_i_object,
        after_finform,
        after_subcomp,
        end_of_sentence
```

```
  };
}

class ADVERBIAL_PREPOSITION {
  public:
    place_in_sentence::PLACE_IN_SENTENCE place;
    vector<OBJECT> objects;
    vector<PREPOSITION> prepositions;

    int begin_pos, end_pos;
    RELATION relation;

    void add_object(OBJECT object);
    void add_preposition(PREPOSITION preposition);
    ADVERBIAL_PREPOSITION();
};
```

`place` is a `PLACE_IN_SENTENCE` enumeration and defines the place where the adverbial was placed in the original sentence. Output modules will try to keep this place, but sometimes it is impossible, because some languages have restrictions where which adverbials are placed.

Of cource `place` has no meaning if the adverbial is child of an object.

4.10 PREPOSITION

The `PREPOSITION` class is only used as child of `ADVERBIAL_PREPOSITION`. It represents, like the name suggests, a preposition.

```
class PREPOSITION {
  public:
    string preposition;
    RELATION relation;
    PREPOSITION(string preposition="");
};
```

`preposition` is a simple string containing the preposition translated in Esperanto. The original preposition is omitted as there are so few prepositions.

4.11 PREDICATE

A `PREDICATE` contains the full predicate of the sentence.

```
class PREDICATE {
  public:
    vector<VERB> verbs;

    int time;
    int person;
    bool plural;
    bool perfect;
    bool passive;
```

```
        bool imperative;

        int begin_pos, end_pos;

        VERB last_verb() const;
        bool exists() const;
        void add_verb(const VERB &verb);
        PREDICATE();
    };
```

`verbs` All verbs belonging to the predicate. The first verb is the finite form.

`time` time, can be 1 (present), 2 (past) or 3 (future)

`person` person, can be 1 (1st person), 2 (2nd person) or 3 (3rd person)

`plural` is true if the finite form is plural

`perfect` is true if the finite form is perfect

`passive` is true if the finite form is passive

`imperative`
is true if the finite form is an imperative

`VERB last_verb()`
returns the last verb

`bool exists()`
returns true if there is minimum one verb, i.e. if the predicate exists

See the explanation for `begin_pos` and `end_pos` in the `SUBJECT` section.

4.12 VERB

This class represents one verb of the predicate.

```
    class VERB {
      public:
        string verb;
        string verb_orig;
        bool verb_known;

        int begin_pos, end_pos;
        RELATION relation;

        VERB();
    };
```

4.13 RELATION

A `RELATION` is an enumeration type which indicates the relation of a subject, object, adjectival or other phrase with its predecessor.

```
enum RELATION {
  r_none,
  r_and,
  r_or,
  r_but,
  r_but_also
};
```

Imagine the following sentence: "The ugly and dirty cat sees your father." In this sentence, the second adjectival has the relation `r_and`, and the first adjectival has the relation `r_none`.

4.14 More Subjects in One Sentence?

As you can see, SENTENCE contains vectors for subject, direct object, indirect object and subject complement. This means that it is possible to have e.g. more than one subject within one sentence.

In normal linguistics, it is impossible to have more than one subject in one sentence. The subject can have more than one part, but it is still one phrase:

"*The big cat and the small dog are in the room.*"

Normally there would be only one subject in this sentence: "The big cat and the small dog". In Transtalo, however, there are two subjects: "The big cat" and "the small dog". This is because the subjects all have their own articles, adjectivals and numbers (single or plural). The two subjects are totally separated from each other.

Let's take another, different, example:

"*The big cats and dogs are in the room.*"

Transtalo treats this as only one subject, just like in normal linguistics. This is because here the nouns ('cats' and 'dogs') are not separated—the article ('the') as well as the adjectivals ('big') belong to both nouns.

Both forms can be combined, too:

"*The big cats and dogs and a yellow rabbit are in the room.*"

There are two subjects in this sentence: "The big cats and dogs" and "a yellow rabbit".

5 XML Sentence Description Files

Besides the `SENTENCE` class, Transtalo also uses XML to describe sentences.

The input and output modules use them to communicate with each other, and the user can use them by making a sentence entirely in XML format to let an output module translate it.

In this chapter, the full format of these files is described.

You might want to have a sample XML sentence file while reading this chapter. You can easily generate it by translating a sentence with Transtalo:

```
transtalo lang2xml eo "^Cu en la arbo la grandaj knaboj vidas min tre bone?"
```

This command will generate an XML sentence file with the sentence "Do the big boys see me very well in the tree?" and put it into '`sentence.xml`' (the default value).

5.1 XML Header

The sentence file begins with an XML header:

```
<?xml version="1.0"?>
```

5.2 Toplevel Sentence

After the header, the first tag is the <sentence> tag. It fills up the rest of the file and represents the whole sentence.

`<original-lang>`
: The original language code (existing of 2 or 3 letters) is between these tags. Should be omitted if it is not applicable (e.g. when you create the XML sentence file manually).

`<original-sentence>`
: The original sentence before the translation. Should be omitted if it is not applicable (e.g. when you create the XML sentence file manually).

`<subject>`
: Represents one subject in the sentence, see the Object section.

`<d-object>`
: Represents one direct object in the sentence, see the Object section.

`<i-object>`
: Represents one indirect object in the sentence, see the Object section.

`<subcomp>`
: Represents one subject complement in the sentence, see the Subject Complement section.

`<predicate>`
: Represents the predicate of the sentence, see the Predicate section.

`<adverbial-preposition>`
: Represents one adverbial with preposition and object in the sentence, see the Adverbial Preposition section.

`<adverbial-adverb>`
: Represents one adverbial with adverb in the sentence, see the Adverbial Adverb section.

`<negative />`
: Is defined if the sentence is negative, e.g. “Low-flying pinguins are not funny!”

`<asking />`
: Is defined if the sentence is asking, e.g. “Does that provide perspectives?”

5.3 Object

This node represents a subject, direct object, indirect object, object belonging to a subject complement or object belonging to an adverbial.

The tag for an object can be `<subject>`, `<d-object>`, `<i-object>` or `<object>`. It has a mandatory argument `type`, it can be one out of:

`type="noun-object"`
: An object with a noun and possible adjectivals, adverbs, etc.

`type="pers-pronoun"`
: An object with a personal pronoun

`type="ind-pronoun"`
: An object with an independent indicating pronoun

Only for `type="noun-object"`:

`<noun>` The noun in Esperanto; has argument `unknown="true"` if the word couldn’t be translated by the input module. Mandatory.

`<original-noun>`
: The noun before it was translated in Esperanto by the input module. Optional.

`<definite />`
: Defined if the noun is definite, i.e. ‘the’ instead of ‘a(n)’ in English.

`<plural />`
: Defined if the noun is plural.

`<force-no-article />`
: Defined if the output module should not add an article to the object.

`<little />`
: Defined if the noun is in its little form. In most languages this must be translated by adding the adjective ‘little’.

Only for `type="pers-pronoun"`:

`<person>` The person, can be 1, 2 or 3. Mandatory.

`<plural />`
: Defined if the personal pronoun is plural.

`<gender>` Gender, can be 0 (neuter), 1 (male) or 2 (female); only applicatable if person is 2. Optional.

`<definite />`
: Article will be definite if there is an article due to presence of an adjectival or adverbial.

`<polite />`
: Defined if the personal pronoun is polite, e.g. the German 'Sie' instead of 'du'.

Only for `type="ind-pronoun"`:

`<ind-pronoun type="type" />`
: Tag for the indicating pronoun. Type can be `this` or `that`.

For all `type`s:

`<adjectival>`
: Represents an adjectival, see section Adjectival

`<adverbial-preposition>`
: Represents an adverbial with a preposition and an object, see section Adverbial Preposition

`<ind-pronoun type="type" />`
: Represents an indicating pronoun. Type can be `this` or `that`.

`<original-position begin="begin" end="end" />`
: Defines the position of the phrase in the original sentence before the translation.

5.4 Subject Complement

The `<subcomp>` tag has one mandatory argument: `type`, which can be one of the following:

`type="adjectivals"`
: The subject complement consists of one or more adjectivals, e.g. "My cat is blue."

`type="object"`
: The subject complement consists of an object, e.g. "This is a nice green cat."

The node consists of one or more `<adjectival>` tags or an `<object>` tag and can contain the `<original-position>` tag, which was already described at the end of the Object section.

5.5 Predicate

The `<predicate>` tag contains the full predicate of the sentence. It has the following tags:

`<time>` The finite form's time is between these tags. 1=present, 2=past, 3=future.

`<person>` The finite form's person is between these tags. Can be 1, 2 or 3.

`<plural />`
: Defined if the finite form is plural.

`<passive />`
: Defined if the finite form is passive.

`<imperative />`
: Defined if the finite form is an imperative.

`<perfect />`
: Defined if the finite form is perfect (i.e. finished).

`<verb>` Represents one verb. Normally the first one is the finite form. See the Verb section for this.

5.6 Verb

The `<verb>` node represents a verb and is always a child node of `<predicate>`.

`<verb>` Verb in Esperanto is between these tags. It has the argument `unknown="true"` if the verb was not known by the input module.

`<original-verb>`
: Original verb before the translation is between these tags. Optional.

`<original-position />`
: See the Object section for this tag.

5.7 Adjectival

The `<adjectival>` node represents, such as the name suggests, an adjectival.

`<adjective>`
: The adjective in Esperanto is between these tags. It has two arguments: `degree="degree"`, which can be 0 (1st degree), 1 (2nd degree) or 2 (3rd degree); the second argument is optional and is `unknown="true"` which is defined if the adjective wasn't known by the input module.

`<original-adjective>`
: The original adjective before the translation is between these tags. Optional.

`<original-position />`
: See the Object section for this tag.

5.8 Adverbial Preposition

The `<adverbial-preposition>` node represents an adverbial that consists of one or more prepositions with one or more objects.

`<preposition>`
: Preposition in Esperanto between these tags. Because there exist so few prepositions, the known and original things are omitted. There may be more prepositions within one adverbial preposition.

`<object>` Represents one object. See the Object section for this.

`<original-position />`
: See the Object section for this tag.

5.9 Adverbial Adverb

The `<adverbial-adverb>` node represents an adverbial that consists of an adverb.

`<adverb adverb="adverb" original-adverb="original adverb" degree="degree">`
: The adverb. Degree is 0 (1st degree), 1 (2nd degree) or 2 (3rd degree).

`<original-position />`
: See the Object section for this tag.

6 Programming Input Modules

To be written.

7 Programming Output Modules

This chapter is a step-by-step guide that tells how to transform the output module template in a translation module and will also give some hints about programming the actual translation progress.

7.1 Introduction

Output modules are translation modules for the Transtalo Translation System that have to perform the translation to a human language. They go futher where input modules stop: the translation from a human language.

Output modules are simply binaries that accept command line options. They output their results to standard output.

Output modules are specialized in a single language: the target language. The only thing they have to do is translating a SENTENCE object into a real sentence.

7.2 Contacting the Project Maintainer

The first thing you need to do is contacting the project maintainer. You can do that by joining the transtalo IRC channel (#transtalo at irc.oftc.net) or contacting via email (tijmen@users.sourceforge.net).

7.3 Obtaining the Output Module Template from CVS

There is a template available for output modules in CVS. To use it, check it out in a directory:

```
$ cd DIRECTORY
$ cvs -d:ext:username@cvs.sf.net:/cvsroot/transtalo co templates/output-template
```

The result will be in templates/output-template. (If you aren't a Transtalo developer, use -d:pserver:anonymous@cvs.sf.net:/cvsroot/transtalo.)

The next thing you do is renaming the dir into transtalo_output_xx (with xx changed into the language code) and removing all CVS subdirectories:

```
$ mv output-template transtalo_output_xx
$ cd transtalo_output_xx
$ rm -rf $(find | grep /CVS)
```

7.4 Understanding the Template

The src/ subdirectory already has five source files:

'`main.cc`' Here does everything begin. This file is rarely changed. It parses the command line options, checks the dialect (if applicable), changes to the directory of the executable (to safely access data files), parses the XML sentence file, and calls the translate() function.

'`translate.cc`'
Here is the function that does the real translation progress. It is nearly empty as you get it.

‘idiom.cc’
: Here come the functions that change the idiom things from the source language into idiom of the destination language.

‘diagnostic.cc’
: Here are useful functions that output diagnostic messages: errors, warnings, and information messages. It automatically checks the verbosity: if verbosity is verbose, all messages are shown; if verbosity is quiet, no messages are shown; if verbosity is normal, only errors and warnings are shown.

‘prefix.cc’
: This is binreloc from autopackage, to find out the directory where the module resists.

You will create more source files that are specialized in one or some language elements.

7.5 Changing the Template

After getting the template, you need to make it ready for your language. Perform the following steps:

- Change the copyright information in all source files: Main author becomes your name and update the year if needed.
- ‘/configure.ac’: transtalo_output_xx: change xx in your language code
- ‘/README’: update this file
- ‘/TO-PROGRAMMERS’: remove this file
- ‘/AUTHORS’: add your name
- ‘/share/output_xx.xml’: rename this file and edit the contents
- ‘/share/Makefile.am’: change output_xx.xml with your language code
- ‘/src/diagnostic.cc’: change xx in all output_xx instances
- ‘/src/main.cc’: change xx in all output_xx instances; change the copyright, etc.
- ‘/src/Makefile.am’: change xx in output_xx
- Run aclocal, autoconf and automake
- Run ./configure and try to make it.

7.6 Storing It in CVS

Do the following to store your module in CVS for the first time (i.e. importing it):

```
$ cd transtalo_output_xx
$ make distclean
$ cvs -d:ext:username@cvs.sf.net:/cvsroot/transtalo import \
    transtalo_output_xx vendor start -m "Initial import of output_xx"
```

7.7 Getting Information From Files

To be written