UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Computer Science

**Helen Kask**

# Pseudolocalization Realization for Skype Software

Bachelor's Thesis (6 EAP)

Supervisor: Krista Liin

Author: ..... "....." May 2012

Supervisor: ..... "....." May 2012

Approved for defence

Professor: ..... "....." May 2012

TARTU 2012

# Table of Contents

# Introduction

In order to be successful in a modern global marketplace, companies have to emerge to other countries and cultures than they originally come from. To succeed in this, the product needs to be localized - adapted to meet the standards of the target market. Localization involves translation as well customization related to different number formats, graphical imagery etc.

Localization is carried out at the end of the development process in order to avoid the need to re-localize due to changes in the design or text. Therefore all mistakes found during the localization process come with a high price. Common problems are text expansion in translated texts compared to the original (especially if the source text is in English), hard-coded text, character encoding problems, format issues, unsuitable imagery etc.

One of the methods to foresee localization problems is pseudolocalization. Pseudolocalization is a practice of simulating the localization process before the actual translation work by replacing the translatable text with modified text [1]. Different pseudolocalization methods address various localization problems including text expansion, hard coded strings, concatenated strings, encoding problems, etc.

The goal of the present thesis is to create a pseudolocalization application suitable for Skype software that would help to streamline the localization and testing process by the Skype localization team.

Chapter one gives an overview of the essence of localization and its biggest problems. It also introduces different pseudolocalization methods and explains what localization problems they help to prevent and/or solve.

Chapter two focuses on the current available solutions for pseudolocalization and analyses their positive and negative sides.

Chapter three describes the process of creating the pseudolocalization application for Skype – which methods were chosen and why and how they were implemented.

Chapter four analyses the results of the application and gives future development possibilities for it.

The appendices contain the sample input file for the program, the program code and a CD containing the executable jar file of the program as well as the digital copy of the thesis.

# 1. Introduction to Pseudolocalization

## 1.1 What is localization?

Significant amount of the software developed nowadays is not meant to be used only in one culture and country. In order to increase user base and revenue, companies have to emerge to new markets other than their original location. Some software companies still debate whether it makes sense to customize their services and assume that potential buyers "probably speak English", but a research conducted by Common Sense Advisory, one of the market leaders in globalization research, indicates that people prefer buying in their own language – 56.2 per cent of consumers say that the ability to obtain information in their own language is more important than the price [2]. Therefore, customizing software for a specific market has become a part of the development cycle for the majority of bigger software companies. The standard process for creating globalized software includes internationalization, which covers generic implementation and design issues, and localization, which involves translating and customizing a product for a specific market [3].

Localization is the process of adapting a product, an application or a document content to meet the linguistic, cultural and other requirements of a specific target market (a locale) [4]. Localization (often abbreviated as L10n, 10 standing for the number of letters between l and n) is frequently used as a synonym for translation, but it entails significantly more processes than the latter. Localization can also involve (but is not limited to) the customization related to numeric, date and time formats, keyboard and currency usage, sorting lists, symbols, icons and colours or even references to certain ideas or objects which, in a given culture, may be subject to misinterpretation or viewed as insensitive [4].

Localization goes hand-in-hand with internationalization (abbreviated as i18n) – the design and development of a product, application or document content that enables easy localization for target audiences that vary in culture, region, or language [4]. Internationalization is a part of the development process and only when the product is ready, it is localized. This is because localization is a fairly expensive process, due to the need for human translators and project managers, and when the product is not finalized yet, localization might have to be done again due to the changes in the user interface (UI). The change may occur either in the layout or English source text etc.

One of the most essential tasks of internationalization is separating localizable elements from source code or content [4] and creating the language file – a plain text file containing only the translatable text elements of the source language, extracted from the source code. Usually, each translatable string is assigned a unique key, and the language file contains key-value pairs. Additionally, the language file may include comments about the source string (its location and usage in the UI, noun/verb distinction etc.) as well as translation length limits.

Other issues to take into account for software internationalization may include enabling the correct character encoding, taking care of concatenation of strings, supporting right-to-left languages and supporting other local preferences such as date and time formats, local calendars, number formats and numeral systems, sorting and presentation of lists, handling of personal names and forms of address, depending on the standards and norms in the target market. [4]

## 1.2 Common problems in localization

Localization is an error-prone process, though most of the problems arise from an incomplete internationalization process. Firstly, translated text can be significantly longer than source text, especially if the source text is English. Some languages expand up to 70 per cent and certain words can expand to 300 per cent [5]. See the example below of how "view profile" is translated in Skype:

*English:* *View Profile*
*French:* *Afficher le profil*
*Russian:* *Посмотреть личные данные*

Text expansion can result in a broken UI if the increased space requirements have not been taken into account during development and design.

Secondly, it is often not considered that different cultures use different formats for displaying various data such as date (for example dd/mm/yyyy instead of mm/dd/yyyy) and time (e.g. 24-hour clock instead of AM/PM format), currencies, numbers, addresses and telephone numbers [5].

Another set of problems arises from the language file and its design. Text in graphics can be difficult to extract, therefore it should be avoided in the first place [5]. Any part of the text should not be hard coded, since it can result in text not being sent into translation; at the same time it is best practice to keep away all text not meant for translation from the language file to avoid any possible functional issues [5]. Concatenating strings i.e. putting longer messages together from shorter strings and reusing the substrings may lead to nonsensical results in translations, since the word order and general sentence structure varies greatly from one language to another [5].

A whole separate category is culturally appropriate imagery and language, which can cause very embarrassing results for the company [5]. For example, the "thumbs-up" sign (👍) used as a positive gesture in the Western world to approve something, is considered very offensive in Latin America and West Africa, Middle-East, as well as Greece, Russia, Sardinia and the south of Italy, where the thumbs-up basically means the same as showing the middle finger in the Western culture [6].

Finally, there are character encoding and right-to-left layout related problems, mainly coming from the fact that the needed scripts are not supported in the software. This can for example

result in the application's UI showing question marks instead of actual characters and right-to-left languages (e.g. Arabic and Hebrew) aligned inaccurately to the left.

Localization stands usually at the very end of the development cycle, therefore all mistakes found during localization can be fixed only with a high price. Depending on the scope of the occurred problem, quite often either redevelopment of the UI is needed, which takes time and delays the release, or the company needs to accept a bad localization solution which is possibly not very suitable for the target market. Both of these options are far from ideal, therefore all methods, which help to prevent or foresee any type of problems related to localization, are extremely useful. One of these methods is pseudolocalization.

## 1.3 Pseudolocalization methods

Pseudolocalization is the practice of simulating the localization process before real translation work – that is, replacing the localizable text within the material in question with something that is easily identifiable in the UI [1]. This can be carried out in many different ways either via a script, macro etc. The main goal of pseudolocalization is to manipulate the language file automatically without the actual human translator, and the results have to be analyzable for a non-native speaker of the target language [1]. Pseudolocalization methods are mostly used to prevent problems regarding text-expansion, character encoding [1] and non-localizable text resulting from hard-coded text or missing text files [7].

One of the most straightforward methods of pseudolocalization is replacing all characters within a string with an "X" or similar easily-identifiable character. This is an easy way to find hard-coded texts or identify text files that have not even made it to the localization process [1].

Two methods that also allow estimating problems caused by text expansion are *Pseudoese* and *Lorem Ipsum*. *Pseudoese* is a fictional written language where Roman characters get replaced by similar-looking diacritical characters, where all vowels are pairs of diacritical vowels, which expands the text. For example, "Bookcase" would look as "Böøöøkcääsêë" [1]. The advantage of using *Pseudoese* is that the source text is still readable, but easily identifiable as being localized [1]. *Lorem Ipsum* is similar to the first method, but is designed to emulate the look-and-feel of actual text, since the text is replaced by pseudo-Latin text [1], which has a more-or-less normal distribution of letters and also expands the text field [8].

In order to analyse where a text has come from and where strings have been concatenated, there are other methods such as adding string identifiers and suffixes and prefixes to the string [1]. Adding an identifier to the string helps to identify where a text in the UI is coming from in the language file [1]. Adding certain easily distinguishable prefixes and suffixes to the string, such as X_string_X can help to see where the string are prone to get cut off on the screen [1] and to see where the strings have been concatenated [7].

Probably the most sophisticated pseudolocalization method is running the text through a machine translator, this is because this method is closest to what will actually happen in the translation process [1]. Machine translation indicates non-Latin alphabet specific issues (e.g. when translating into Chinese), it brings out character encoding and text-expansion problems as well.

The best practice is to use different pseudolocalization methods in combination to cover a wider range of problems. The features of different pseudolocalization methods have been presented conclusively in Table 1 below.

**Table 1. Comparison of pseudolocalization methods.**

| | **Hard-coded strings** | **Text expansion** | **Character encoding** | **Right-to-Left** | **Concate-nation** | **String source detection** |
|---|---|---|---|---|---|---|
| **X replacement** | Yes | No | No | No | No | No |
| ***Pseudoese*** | Yes | Yes | Yes | No | No | No |
| ***Lorem Ipsum*** | No | Yes | No | No | Yes | No |
| **Prefix/Suffix** | Yes | Yes | No | No | Yes | No |
| **String ID** | Sometimes | Sometimes | No | No | Yes | Yes |
| **Machine translation** | Sometimes | Yes | Yes | Yes | No | No |

# 2. Available Solutions for Pseudolocalization

## 2.1 Public code libraries for pseudolocalization

All the available solutions that could be found online for pseudolocalization fall under two categories: freely available code libraries that give you a start point for developing your own pseudolocalization application, or sophisticated commercial tools made for the entire localization process.

Google pseudolocalization-tool includes a structured message API to allow it to be used for complex multi-part messages, and includes 4 pseudolocalization methods [9]. These are "accenter", which replaces ASCII characters with their accented version, a version of *Pseudoese*; "brackets" which is an example of the suffix/prefix method adding square brackets ([ ]) around the string; "expander" which just makes the message longer and "fakebidi" which produces fake right-to-left text [9]. The API gives a good structure if you want to develop a larger scale pseudolocalization application. It has classes for html text, its localizable and not-localizable parts. The "fakebidi" method is very useful for testing Right-to-Left properties of the software.

Localization Tools Maven Plugin is a simple plugin within Maven - a software project management and comprehension tool. Maven Plugin allows choosing which sources to pseudolocalize and which to exclude and implements the prefix/suffix method [10]. Unfortunately, this only works for projects developed using Maven, which is a big limitation, since usually localization and therefore pseudolocalization is only considered when the project is either ready or almost ready and project management tools already chosen.

Pseudolocalizer class posted to MSDN library by David Anson implements the prefix/suffix and *Pseudoese* methods, but only works for standard .NET RESX-based resources [11], which is a big limitation on the input file types.

The "pLoc" tool posted to the Code Project provides a command line and simple user interface solution for pseudolocalizing input text using prefix/suffix and *Pseudoese* methods [12]. It is written in C# and it only accepts plain text – the text that needs to be localized, not a line from an XML file for example.

All these solutions have one big common downside. They are all far from a ready-made solution and still require the actual pseudolocalization application to be developed from scratch, except the Maven plugin, and they do not include any solution for machine translation. Also, most of them have strict restrictions for the input.

## 2.2 Applications supporting the entire localization process

Crowdin is an online tool to be used for the entire localization process. It allows project management for uploaded files, using previous human translations in the online translation memory, creating a community to translate projects into any language needed and having the

project synchronized online between all parties [13]. Crowdin also has built-in pseudolocalization methods for *Pseudoese* and suffix/prefix methods as well as machine translation [13].

Alchemy Catalyst and SDL Passolo software are both desktop applications and therefore have even more functionality than Crowdin. These are related to translation memory (a database for keeping translations so they could be reused) functionality as well as project management and supporting more file formats. Both of these applications support machine translation and they also allow rendering the pseudolocalized text in the UI for certain file types [14], [15].

These higher level tools suit perfectly for projects in the beginning phase that have not set up their localization processes yet. Crowdin is a cheaper and simpler tool, Alchemy Catalyst and SDL Passolo offer more features but are more expensive as well. Compared to Alchemy Catalyst and SDL Passolo, it is fairly easy to use only the pseudolocalization feature in Crowdin without having to use any of the other features.

On the downside for the mentioned tools, if the organization has already set up the general localization process and is not interested in changing it, there is no option of just using the pseudolocalization component and it is not cost-efficient to pay only for a small part of the complete functionality of the product. They are also fairly uncomfortable to use if one is only interested in pseudolocalization due to the project management overhead (even though the project management related features would be useful if the tools were used for their entire purpose).

Lingoport Globalyzer is different from the above mentioned tools, since it focuses on the internationalization part of the process and works directly with the code without providing functionality for localization processes. It provides a full code analysis and editing environment for finding, fixing, testing and reporting the internationalization issues in a wide variety of programming languages [16]. Therefore the Lingoport Globalyzer deals with the same problem as pseudolocalization methods in general – to prevent localization problems in the software in advance – but does it in more sophisticated ways. Globalyzer can be run as a stand-alone desktop editor or as an Eclipse™ Plugin [16]. On the good side, the Lingoport Globalyzer provides full internationalization testing. On the downside, it is not free of charge and requires changes to the existent development process and toolset.

## 2.3 Evaluation of existing solutions

To sum up, there are plenty of good pseudolocalization tools available if the software development project is started from scratch. This means being able to easily implement different tools into the processes, choose the file types according to the tools available and being willing to pay for the solutions. In other cases, probably a customized pseudolocalization tool has to be built, but the help of existing APIs can be used in this case.

The features supported by different pseudolocalization solutions are presented conclusively in Table 2 below.

**Table 1. Comparison of pseudolocalization tools.**

| | ***Pseudoese*** | **Prefix/ Suffix** | **Machine translation** | **Complete toolset** | **Online/ Desktop** | **Free** |
|---|---|---|---|---|---|---|
| **Google API** | Yes | Yes | No | No | Desktop | Yes |
| **Maven Plugin** | No | Yes | No | No | Online | Yes |
| **pLoc** | Yes | Yes | No | No | Desktop | Yes |
| **Anson** | Yes | Yes | No | No | Desktop | Yes |
| **Crowdin** | - | - | - | Yes | Online | No |
| **Alchemy Catalyst** | - | - | - | Yes | Desktop | No |
| **SDL Passolo** | - | - | - | Yes | Desktop | No |
| **Lingoport Globalyzer** | - | - | - | Yes | Desktop | No |

# 3. Pseudolocalization application for Skype

## 3.1 Introduction

Skype is a software for audio and video calls and instant messaging between Skype users as well as making phone calls to mobiles and landlines [17]. Skype was first released in 2003 and is now supported on all major consumer platforms including Windows, MacOS, Linux, iOS, Windows Phone, Android as well as TV and gaming platforms [17]. Skype is used up to 40 million people concurrently at the same time [18]. Skype is localized into up to 32 languages depending on the platform and at least half of Skype users prefer to use Skype in some other language than English. Skype localization is handled by a central localization team cooperating with development and other teams whose content needs localizing.

Even though Skype clients running on different platforms are constantly being improved, most of them have been already developed some time ago and localization standards have been set. Therefore some of the common problems in the localization process hardly ever occur in Skype, such as problems with character encoding or a language file that has not fully been separated from code. One thing that can rarely be predicted though is text expansion. This is especially problematic on mobile applications, where space limitations are very strict and it is not easy to leave some extra room in the UI design just in case. This is why having a good pseudolocalization application accurately describing text expansion could be of great help for Skype.

Skype has established its localization and development processes long ago with more than 20 languages supported as far back as 2006 [19]. Therefore, applications supporting the entire localization process are not suitable solutions for pseudolocalization for Skype, for reasons described in section 2.2. The available code libraries do not support machine translation and have usually very strict constraints on the input files, which is why it was decided to develop a separate simple Java application for pseudolocalization using Microsoft Translator. Java was chosen due to multiplatform support and the developer's background in it.

## 3.2 Functionality

### 3.2.1 Pseudolocalization methods

The pseudolocalization application for Skype implements machine translation method using Microsoft Translator, as well as two other additional pseudolocalization methods: prefix/suffix and string identifier.

The most accurate pseudolocalization method for text expansion is machine translation. This is particularly useful in cases, where space limitations are very strict and it is necessary to know rather accurately how much space it is absolutely necessary to leave for the content. *Pseudoese* and prefix/suffix method always give the same percentage of expansion,

independent of the target language. In contrast, depending on the language, machine translation results can be very accurate.

Microsoft Translator was chosen as the machine translation solution, since Google Translator – the other one of the two biggest machine translation solutions – has closed its public API. Microsoft Translator results for translating all the Skype for Windows UI strings from English to German were analysed against the existing official human translations. The results were 100% accurate length-wise on more than half the cases for short source texts under 10 characters. This is an extremely good result, given that the translations of short English source texts are the hardest to predict expanding up to 300% in translation [20].

Even though machine translation is the most sophisticated solution for predicting text expansion, in case of languages sharing the same type of alphabet (e.g. Latin for most European languages ), it doesn't give a clear concise overview, whether all the English source text has made it to the language file (i.e. there are not any hard-coded strings). Even though such hard-coding does not occur often in Skype, it is still useful to have a quick way of getting a clear overview if all text has made it to the language file when new features are being added to the Skype client. Furthermore, machine translation does not show anything about string concatenation. This is why it was also decided to add the prefix/suffix method which is of great help in locating concatenated strings and hard-coded text.

The method of replacing the string with its identifier (the key from the language file) helps to find quickly where in the language file is the string that causes the problem seen in the UI.

The ideal workflow for using pseudolocalization application for Skype would be first to indicate not translated or concatenated strings with the suffix/prefix method and then check text expansion problems using the machine translation method. The final step would be to locate the problematic strings in the language file with the string identifier replacement method.

### 3.2.2 Microsoft Translator

Microsoft Translator supports translation between 38 languages, including all the 32 languages Skype currently supports. Microsoft Translator API can be used in software and websites using either Web widget, AJAX, HTTP, SOAP, or OData interfaces [21]. AJAX is used for instant translation for web content [22], SOAP can easily be used in Visual Studio (a Microsoft integrate development environment) to use translations inside software [23], HTTP interface allows translation functionality using HTTP GET & POST methods, which do not require any specific operation system or programming language to develop using it [24]. HTTP GET & POST methods were chosen for pseudolocalization application for Skype due to the independence from the programming language.

The usage is free of charge if the API is used for translating less than 2 million characters a month [25]. This limit is enough for translating additions to the UI, since software generally does not contain too much text. Should somebody want to use Microsoft Translator for

testing a webpage layout with more text, then possibly bigger limits need to be purchased. For using Microsoft Translator API, an access token has to be obtained, which is used for authentication and passed on with each API call [26]. The access token expires after ten minutes and has to be renewed [26]. This means that when an application runs more than ten minutes, then the token has to be renewed. This is not a problem for the pseudolocalization application for Skype since it never runs this long.

### 3.2.3 Input and output of the application

Skype software is originally in English therefore the source language for localization process is English. The language files used for Skype software vary in size and structure depending on the platform. For example, Skype for Windows language file is very simple, with each line containing key-value pairs. It contains nearly 3000 strings, whereas other Skype clients' language files are much smaller. An example from the Skype for Windows language file, where the key is on the left and translatable string on the right:

*sSTATUSMENU_CAPTION_ONLINE=Online*
*sSTATUSMENU_CAPTION_OFFLINE=Offline*

Most of Skype clients' language files can be parsed into and from an XML filetype used internally at Skype, which is why this file is used as the input and output for the application. This means that the XML output from pseudolocalizer can be parsed back to the native language file, which can then be loaded to the Skype client as any new language file and checked for problems. The XML file contains a header and for each content line the key, comments, translation length limit and the translatable string itself as it can be seen below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bundle xmlns="http://integration.skype.net/wtt/bundle/1.0/">
  <meta>
    <project-name>test1</project-name>
    <component-name>test</component-name>
     <country-code>US</country-code>
     <language-code>EN</language-code>
     <file-id>117100000069055370</file-id>
    <file-name>117100000069055370_en_US_2012-04-
24_170942+0000.xml</file-name>
    <created-by>helen.kask</created-by>
    <date-created>2012-04-24T17:09:42.345Z</date-created>
  </meta>
<content>

      <item comment="Menu name under Tools - Options - Advanced " key="
sSTATUSMENU_CAPTION_ONLINE " maxlen="5000"
translatable="yes">Online</item>
```

*</content>*
*</bundle>*

The output file needs to have exactly the same structure as the file shown above. The only items changed are the strings themselves, which have been converted by the chosen pseudolocalization method. Appendix 1 contains an example of Skype XML file type with more translatable strings.

### 3.2.3 Implementation

The pseudolocalization application for Skype is a command line program, getting its parameters listed in the command line during the launch. The parameters are the input filename, the output filename, the requested method and if choosing machine translation, then also the output language and input language. The default values are *"input.xml"* for the input file, *"output.xml"* for the output file, machine translation for the method, English for the input language and German for the output language (because German is one of the languages with the most notable expansion compared to the English source). The methods are coded as numbers, 1 standing for machine translation, 2 for suffix/prefix method and 3 for replacing the string with its key. For example launching the application with the statement *"Java –jar Pseudolocalizer.jar"* would launch the application using the default values and *"Java –jar Pseudolocalizer.jar myfile.xml outputfile.xml 2"* would read from myfile.xml, use prefix/suffix method and write to outputfile.xml and *"Java –jar Pseudolocalizer.jar input.xml output.xml 1 ru"* would launch the machine translation from English to Russian reading from input.xml file and writing to output.xml file. When parameters coming later (such as input and output languages) have to be changed, then the previous parameters have to be written out as well.

Graphical user interface was deemed unnecessary, because the users of this program are the localization team members who are experienced and easily trainable and actually using a UI can often take more time than launching the program on the command line reusing previously written statements. Input checking is also not needed, because the XML file is never compiled manually, so the structure will always be correct.

Figure 1 describes the program flow. The program processes the input file line by line with the *read* method. The header (first 13 lines of the input file) is written to the output stream with no changes. Other lines are sent one by one to a parse method. There are 3 different parse methods depending on the chosen pseudolocalization method. The parse method uses standard Java string methods (such as *scanner* that splits text by a delimiter character) to extract the key and translatable part of the line. Depending on the chosen pseudolocalization method, the parse method returns the same line with the string replaced by the key or by *„X string X"*. In case of the machine translation option, the parser method calls out the *translate* method and returns the input line with the string replaced by the translation returned by the *translate* method. The end of language strings and the start of the footer of the XML file is

identified by the parse method by checking the length of the input line and the footer is written to the output file with no changes. Different parse methods make it very easy to add new custom pseudolocalization methods by just adding a new switch case to the process.

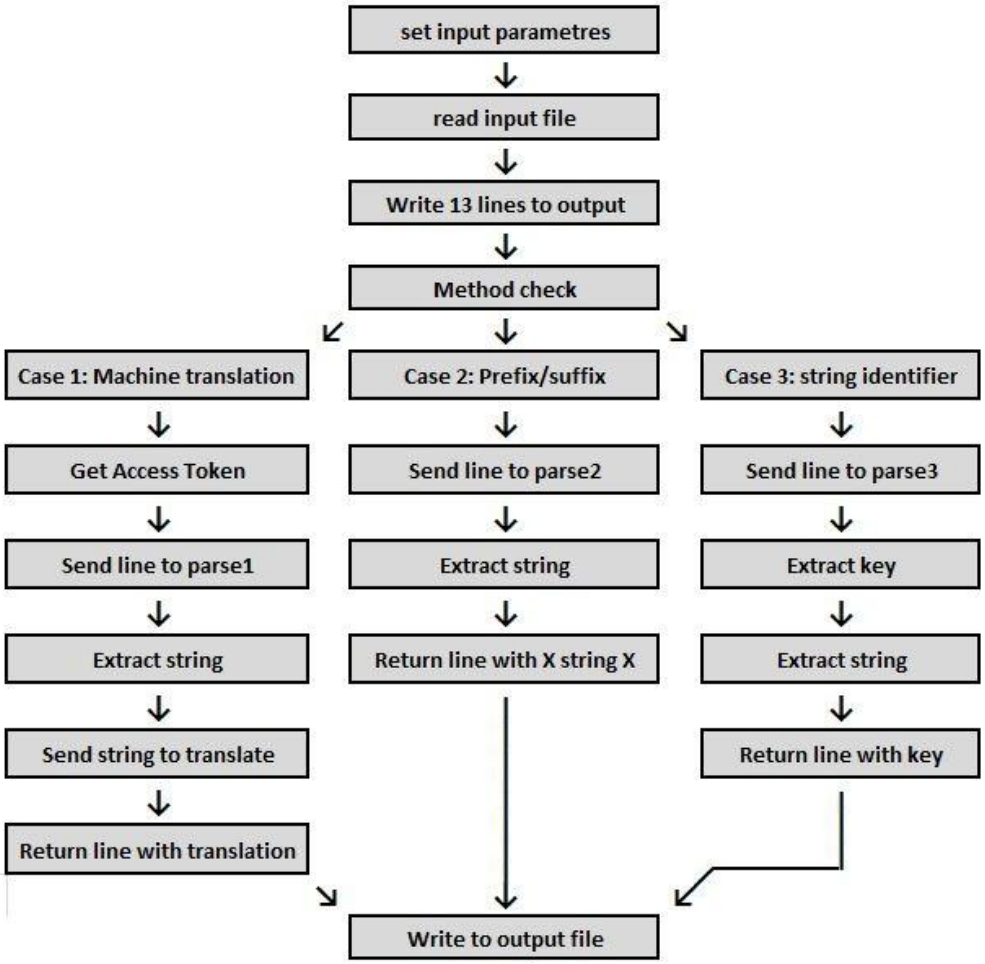


**Figure 1. Flow of the program.**

If the chosen pseudolocalization method is machine translation, then first the *getToken* method is invoked to get the access token, which is used by the translation method to connect to the Microsoft Translator. The *getToken* method uses Java *httpclient* and *httpPost* methods to connect to the Microsoft server. The *httpPost* request passes on client ID and client secret obtained when registering as a developer at Windows Azure Marketplace as well as other info described in Microsoft Translator's webpage ([26]), which the user sets in the constant variables in the program. The access token is valid for 10 minutes and is passed on with each API call (each time the translation method is invoked).

The translate method uses Java *HttpURLConnection* method to make the API call, which passes on the access token. The translation info is passed in the URL the connection is made to, consisting of Microsoft Translator address, the UTF-8 encoded version of the translatable

text and input and output languages. Then the returned XML is analysed in the same way as the lines in the parse method, to get the actual translation from the response.

Implementing *getToken* and *translate* methods was the most difficult and time consuming part of creating the application. This was because the connection method to the Microsoft Translator API was changed in March 2012 and the previous connection method was discontinued. For this reason there were no Java examples available for using the access token and Microsoft only posted examples of C# and PHP [26]. Therefore it was necessary to develop the connection method for Java from scratch using the C# and PHP as a vague reference.

The full program code can be found from Appendix 2 and executable jar file from Appendix 3.

# 4 Results

## 4.1 Application in practice

The pseudolocalization application for Skype helps the Skype localization team to detect problems in the UI in a quicker and more systematic way than previously. The ideal workflow would be first to click through the UI with the language file pseudolocalized with the prefix/suffix method to find concatenated and hard-coded strings, then click through the UI with a machine translated file to see for problems regarding text expansion, and then to use the key method to locate the identified problems quickly.

Potential issues caused by text expansion can be foreseen up to a point by visual checking of the UI by an experienced localization team member, but pseudolocalizing the language file with machine translation is definitely a more sophisticated and quicker solution for that. Below is an example illustrating this – on the left is the image of the English UI and on the right there is the image of machine translation result in Russian. The screenshot is taken of the window shown during screen sharing done during a Skype call, when Skype is running in the background or minimized. As it can be seen in Figure 2, the window is not designed to be scalable depending on the text length. Therefore this can be a potential cause for localization issues, since the machine translation indicates that the translations for this phrase can be rather long. This cannot be detected from the English version.

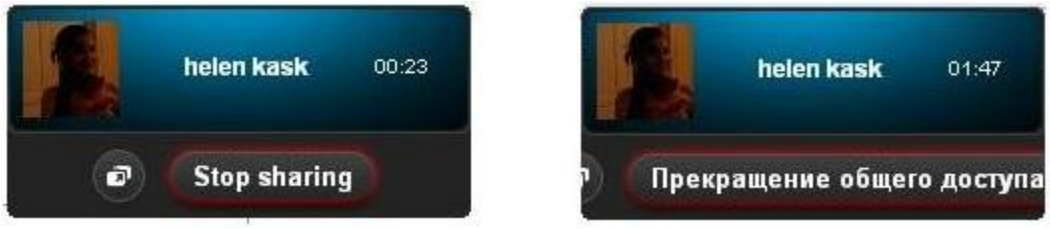


**Figure 2. The string "Stop sharing" English and Russian (machine translated).**

The same applies for prefix/suffix method as well – some of the concatenated strings can be detected by an experienced localization team member by visually checking the language file, but pseudolocalizing the language file with the prefix/suffix method and looking at it in the UI context makes it a lot more difficult to overlook concatenated strings. Prefix/suffix also indicates hard-coded strings rather well, which are quite often only discovered after the actual translations are checked in the UI. Below in Figure 3 is an example of the chat window in English on the left and using the language file pseudolocalized with the prefix/suffix method on the right. As it can be observed in Figure 3, absolutely nothing indicates on the English version that the strings “Call Mobile” and “Call Home” are concatenated, only the prefix/suffix method indicates the potential problem.

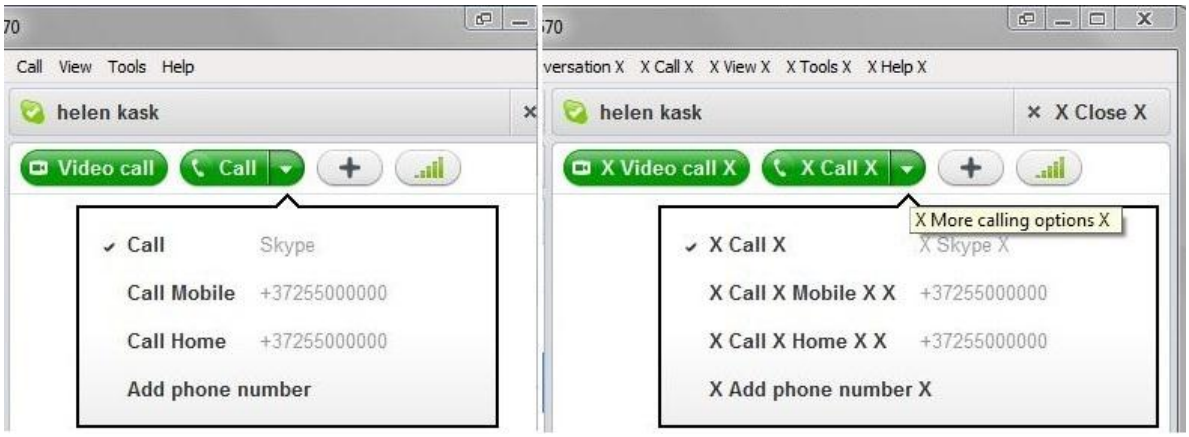


**Figure 3. Call options in English and with prefix/suffix method.**

The pseudolocalization method of replacing the string with its key helps to understand quickly from where in the language file a string is coming from, which is very useful when detecting any of the previously mentioned or other types of problems in the UI. Without this method, finding the right place from the language file can either be done by searching the string from the language file or in the worst cases for more common strings and for a big language file (for example the word "call" which has 865 occurrences in the Skype for Windows language file) by manually replacing the string with something identifiable and then checking whether it changed in the UI or not, which would be very time consuming. Below in Figure 4 is the example of how can this method help to locate the problem which can be seen in Figure 3. As can be seen in Figure 4, the troublesome string can be easily identified and after that one can check in the language file that the problem is the fact that *sCALL_SOMETHING=Call %s*, where *%s* is a variable with its values pulled from another string.

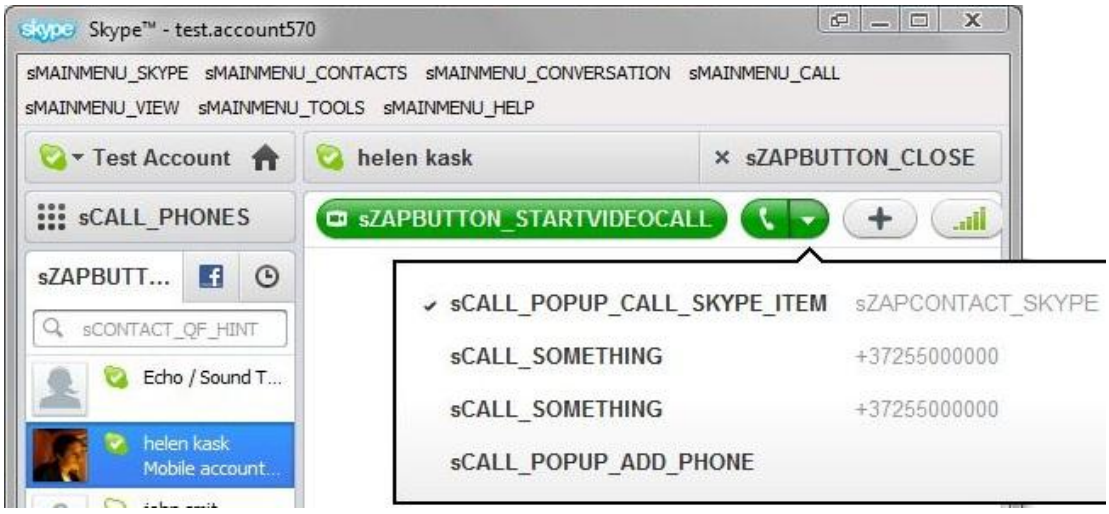


**Figure 4. Call options with keys.**

## 4.2 Evaluation of the application

The goal of creating pseudolocalization application for Skype was to help the localization team to prevent localization problems more easily. The pseudolocalization application for Skype does not save the localization team from clicking through the UI, so it does not make the process any shorter. However, it certainly provides a more systematic way of performing UI review and decreases the amount of problems in the UI that would be overlooked during the review. The deployed methods cover three main categories of localization problems – text expansion, string concatenation and hard-coded text – as explained in section 4.1. Machine translation can also be used to check the support of correct character encoding. Therefore the developed application helps to foresee a wide range of localization problems and discover more problems before localizing the software, which in turn will decrease the time, effort and money needed for solving these problems after finishing the localization process. The application is already in use by the localization team.

The developed application is better from the current existing solutions because it is free of charge and it implements the machine translation method (which the available code libraries do not). The connection to Microsoft Translator in Java can be used for developing other similar applications as well. On the downside, the application only works with the Skype XML file and needs to be redeveloped if it needs to be used for any other file types.

## 4.3 Future development possibilities

With the parse methods being independent methods in the code, it is easy to add new pseudolocalization methods to the program by writing a new parse method. For example, if in the future Skype starts supporting more languages that are not supported by Microsoft Translator, then it is possible to develop a *Pseudoese* language specific to the characters of the new languages to check character encoding problems.

In order to make the machine translation method even more accurate for text expansion, it is possible to make the program take current existing translations into account as well and translate only the parts of the string that have not been translated before. Most current UI translations are available in the described XML format. Creating an optimal search method for this though is a non-trivial task.

The program could also be adjusted to analyse the text expansion in Skype current translations in different languages. This would help to get more accurate estimations for localization team members when reviewing UI plans at a development stage where practicing pseudolocalization is not available yet. It could also be used to analyse the accuracy of Microsoft Translator translation results compared to Skype translations to get a more precise overview on how accurate is pseudolocalization with machine translation, since machine translation results are improved all the time.

A feature for locating black listed terms that are not supposed to be in use anymore in existing translations can be added as well.

# Summary

This work gives a short overview of common localization problems as well as a number of well-known pseudolocalization methods that can be used to prevent those problems. The existing solutions for pseudolocalization were analysed as well. This information can be useful when choosing which solution to use for pseudolocalization or deciding how pseudolocalization can be of help in the development process.

The main goal of the thesis was to develop a pseudolocalization application for solving possible issues in user interfaces for Skype clients. The developed application implements three pseudolocalization methods – machine translation, prefix/suffix and string identifier methods and works with XML file types used internally in Skype. Using these methods helps to prevent localization issues related to text expansion, encoding, hard-coded text and string concatenation as well as helps finding the problematic strings from the language file quicker when using string identifier method.

During the development of the pseudolocalization application for Skype, the connection to the Microsoft translator was developed in Java that did not exist before. This can also be used by other users to connect to Microsoft Translator with Java.

The result of the thesis's practical part is already in use by the Skype localization team.

# Pseudolokaliseerimise realisatsioon Skype’i jaoks

Bakalaureusetöö

Helen Kask

**Resümee**

Käesoleva bakalaureusetöö eesmärgiks on luua pseudolokaliseerimise rakendus Skype’i jaoks. Lokaliseerimine on tarkvara kultuurile kohandamine, mis hõlmab endas nii tõlkimist kui ka numbriformaatide, piltide jm kohandamist kohalikule kultuurile. Peamised lokaliseerimise probleemid on seotud teksti pikenemisega tõlgetes (eriti kui sisendkeeleks on inglise keel), koodi sisse kirjutatud tekstiga, mis jääb seetõttu tõlkimata, mitmest eri sõnest kokkupandud tekstiga, vale märgikodeerimisega (ingl k. *encoding*), erinevate numbriformaatide ning sobimatute piltidega. Üks viis lokaliseerimise käigus tekkivad probleeme ennetada on pseudolokaliseerimine.

Pseudolokaliseerimine kujutab endast lokaliseerimisprotsessi imiteerimist enne teksti päriselt tõlkesse saatmist. See tähendab tekstifaili tekstiväljade muutmist teatud sobival viisil ja seejärel faili vaatamist kasutajaliideses. Erinevad pseudolokaliseerimise meetodid aitavad ennetada kodeeritud tõlkimata teksti, teksti pikenemise, märgikodeerimise- ja mitmest eri sõnest kokkupandud tekstiga seotud probleeme.

Olemasolevad pseudolokaliseerimise rakendused, mida on ka antud töös hinnatud, ei ole Skype’i jaoks sobilikud. Seega arendati eraldiseisev pseudolokaliseerimisrakendus, mis realiseerib sulumeetodi (sõne asendatakse *“X sõne X”*-ga), võtmemeetodi (sõne asendatakse tema keelefaili identifikaatoriga) ja masintõlkemeetodi Microsoft Translator’i abil (sõne asendatakse masintõlkega). Sulumeetod aitab leida kokku kirjutatud sõnesid ning koodi kirjutatud teksti, masintõlke meetod aitab leida potentsiaalseid probleeme teksti pikenemisega ning märgikodeeringuga seotud probleeme. Võtmemeetod aitab keelefailist kiiresti leida sõne, mis kasutajaliideses probleeme põhjustab.

Loodud rakendus on juba Skype’i lokaliseerimismeeskonna poolt kasutuses ning probleemide ennetamisega aitab see kokku hoida probleemide hilisemaks lahendamiseks kuluvaid ressursse.

Muu hulgas programmeeriti töö käigus ka ühendus Microsoft Translator’iga programmeerimiskeeles Java, mis varem puudus, ning mida on võimalik ka teistel kasutada.

# References


[1] Translating Website Content: The Benefits of Pseudo-Localization Techniques. http://blog.lionbridge.com/translation/2009/12/17/translating-website-content-the-benefits-of-pseudo-localization-techniques/. Last visited 20.04.2012

[2] Donald A. DePalma, Benjamin B. Sargent, Renato S. Beninatto, Can't Read, Won't Buy: Why Language Matters on Global Websites, Common Sense Advisory, 2006. http://www.commonsenseadvisory.com/AbstractView.aspx?ArticleID=957. Last visited 15.05.2012

[3] Software Internationalization. http://userguide.icu-project.org/i18n. Last visited 18.04.2012

[4] Localization vs. Internationalization. http://www.w3.org/International/questions/qa-i18n. Last visited 17.04.2012

[5] Nitish Singh, Localization Certification Workshop (online material), 2008, copyright © Localization Certification Program. http://rce.csuchico.edu/localize/demo/Introduction%20to%20Localization.pdf. Last visited 15.05.2012

[6] Language Trainers blog. http://www.languagetrainers.co.uk/blog/2007/09/24/top-10-hand-gestures/. Last visited 12.05.2012

[7] Pseudolocalization to Catch i18n Errors Early. http://google-opensource.blogspot.-com/2011/06/pseudolocalization-to-catch-i18n-errors.html. Last visited 19.04.2012

[8] Lorem Ipsum. http://www.lipsum.com/. Last visited 15.05.2012

[9] Pseudolocalization-tool.http://code.google.com/p/pseudolocalization-tool/. Last visited 21.04.2012

[10] Localization Tools Maven Plugin. http://mojo.codehaus.org/l10n-maven-plugin/pseudo-mojo.html. Last visited 22.04.2012

[11] sudo localize & make me-a-sandwich [Free PseudoLocalizer class makes it easy for anyone to identify potential localization issues in .NET applications]. http://blogs.msdn.com/b/delay/archive/2011/01/27/sudo-localize-amp-amp-make-me-a-sandwich-free-pseudolocalizer-class-makes-it-easy-for-anyone-to-identify-potential-localization-issues-in-net-applications.aspx. Last visited 22.04.2012

[12] pseudoLocalizer -- a tool to aid development and testing of internationalized applications. http://www.codeproject.com/Articles/8496/pseudoLocalizer-a-tool-to-aid-development-and-test#_comments. Last visited 22.04.2012

[13] Crowdin collaborative translation tool. http://crowdin.net/ Last visited 22.04.2012

[14] Products - Alchemy CATALYST 9.0. http://www.alchemysoftware.ie/products/-alchemy_catalyst.html. Last visited 23.04.2012

[15] SDL Passolo 2011. http://www.sdl.com/en/language-technology/products/software-localization/sdl-passolo.asp. Last visited 23.04.2012

[16] Globalyzer. http://www.lingoport.com/software-internationalization-products/globalyzer-4/. Last visited 23.04.2012

[17] Skype. http://www.skype.com. Last visited 17.04.2012

[18] The Big Blog. http://blogs.skype.com/en/2012/04/40_million_people_how_far_weve.html. Last visited 12.05.2012

[19] Skype Developer Blog. http://blogs.skype.com/developer/2006/03/localization_tricky_but_import.html. Last visited 29.04.2012

[20] Text size in translation. http://www.w3.org/International/articles/article-text-size. Last visited 28.04.2012

[21] Getting Started with Microsoft Translator. http://msdn.microsoft.com/en-us/library/-hh454949.aspx. Last visited 2.05.2012

[22] Using the AJAX Interface. http://msdn.microsoft.com/en-us/library/ff512385.aspx. Last visited 2.05.2012.

[23] Using the SOAP Interface. http://msdn.microsoft.com/en-us/library/ff512390.aspx. Last visited 2.05.2012

[24] Using the HTTP Interface. http://msdn.microsoft.com/en-us/library/ff512387.aspx. Last visited 2.05.2012

[25] Windows Azure Marketplace. https://datamarket.azure.com/dataset/1899a118-d202-492c-aa16-ba21c33c06cb. Last visited 15.05.2012

[26] Obtaining an Access Token. http://msdn.microsoft.com/en-us/library/hh454950.aspx. Last visited 6.05.2012

# Appendices

## Appendix 1. Skype XML language file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bundle xmlns="http://integration.skype.net/wtt/bundle/1.0/">
  <meta>
    <project-name>Skype for Windows, Client</project-name>
    <component-name>5.9</component-name>
    <country-code>US</country-code>
    <language-code>EN</language-code>
    <file-id>5176000000002762466</file-id>
    <file-name>5176000000002762466_en_US_2012-05-13_135135+0200.xml</file-name>
    <created-by>helen.kask</created-by>
    <date-created>2012-05-13T13:51:35.723+02:00</date-created>
  </meta>
  <content>
    <item key="sSTATUSMENU_CAPTION_ONLINE" translatable="yes"
maxlen="5000">Online</item>
    <item key="sSTATUSMENU_CAPTION_OFFLINE" translatable="yes"
maxlen="5000">Offline</item>
    <item key="sSTATUSMENU_CAPTION_AWAY" translatable="yes" maxlen="5000">Away</item>
    <item key="sSTATUSMENU_CAPTION_DND" translatable="yes" maxlen="5000">Do Not
Disturb</item>
    <item key="sSTATUSMENU_CAPTION_INVISIBLE" translatable="yes"
maxlen="5000">Invisible</item>
    <item key="sREDBUTTON_HINT_HANGUP" translatable="yes" maxlen="5000">End Call with
%s</item>
    <item key="sGREENBUTTON_HINT_CALL" translatable="yes" maxlen="5000">Call %s</item>
    <item key="sBUDDYLIST_ADDING_YOURSELF" translatable="yes" maxlen="5000">You can't add
yourself to Contacts.</item>
    <item key="sCALL_LB2_MISSED" translatable="yes" maxlen="5000">No answer</item>
    <item key="sCALL_LB2_REJECTED" translatable="yes" maxlen="5000">Busy</item>
    <item key="sQUIT_PROMPT" translatable="yes" maxlen="5000">You won't be able to send or
receive instant messages and calls if you do.</item>
    <item key="sAVATAR_MENUITEM_REJECT" translatable="yes" maxlen="5000">Reject</item>
    <item key="sMAINMENU_HELP_MENU" translatable="yes" maxlen="5000">&amp;amp;Help</item>
    <item key="sMAINMENU_HELP_HELP" translatable="yes" maxlen="5000">Get Help: Answers
and Support</item>
    <item key="sMAINMENU_HELP_UPDATES" translatable="yes" maxlen="5000">Check for
Updates</item>
    <item key="sBUDDYMENU_CALL" translatable="yes" maxlen="5000">Call</item>
    <item key="sBUDDYMENU_SENDMESSAGE" translatable="yes" maxlen="5000">Send
IM</item>
  </content>
</bundle>
```

## Appendix 2. Program code

```
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.StringWriter;
import java.io.Writer;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import org.apache.commons.io.IOUtils;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import java.net.HttpURLConnection;
import java.net.URLEncoder;
import java.util.Scanner;

/**
 * @author      Helen Kask <helen.kask@hotmail.com>
 * @version     1.0
 * @since       2012-05-13
 */
public class pseudolocalizer {
        public static void main(String[] args) throws IOException {
                //Initializing the variables
                for(int i=0; i<args.length; i++){
                        switch (i) {
                        case 0: fFileName=args[0];
                        break;
                        case 1: fFileNameOut=args[2];
                        break;
                        case 2: method=new Integer(args[3]);
```

```
                        break;
                        case 3: outputLang=args[4];
                        break;
                        case 4: inputLang=args[5];
                        }
                }
         try {
                        read();
                } catch (IOException e) {
                        e.printStackTrace();
                        log("Unable to read the file");
                }
        }
        /**
         * Gets access token from MS server.
         *
         * Connects to the server using httpPost and MS Azure client credentials.
         *
         * @return String authentication token to be used for getting translations from MS server.
         */
public static String getToken(){    // Create a new HttpClient and Post Header
  //Http connection to server for getting the token
                HttpClient httpclient = new DefaultHttpClient();
                HttpPost httppost = new
HttpPost("https://datamarket.accesscontrol.windows.net/v2/OAuth2-13");
                try {       // Adding all the needed data for server connection and token
                        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();
                        nameValuePairs.add(new BasicNameValuePair("grant_type",
"client_credentials"));
                        nameValuePairs.add(new BasicNameValuePair("client_id", clientId));
                        nameValuePairs.add(new BasicNameValuePair("client_secret",
clientSecret));
                        nameValuePairs.add(new BasicNameValuePair("scope",
"http://api.microsofttranslator.com"));
                        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));       //
Execute HTTP Post Request
                        HttpResponse response = httpclient.execute(httppost);

                        //To get the token value from the output - the original is an XML
                        ByteArrayOutputStream out = new ByteArrayOutputStream();
                        response.getEntity().writeTo(out);
                        String jsonToken = out.toString();
```

```
                        String[] Tokens=jsonToken.split("\",");
                        String Token = Tokens[0];
                        Tokens=Token.split(":\"");
                        Token=Tokens[2];

                        return Token;

                } catch (ClientProtocolException e) {
                        // TODO Auto-generated catch block
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                }
                return "";
}
/**
 * Translates the input string with Microsoft Translator
 *
 * Connects to Microsoft Translator using HttpURLConnection and authentication token
 *
 *
 * @param  inputText The text sent to translation
 * @param  token The authentication token need to to connect to Microsoft Translator
 * @return Translated version of the input text
 */
public static String translate(String inputText, String token) throws IOException{
          String translatableText = inputText;
          String encText = URLEncoder.encode(translatableText, "UTF-8"); //encode the input text
          URL url=new URL("http://www.samplepagedoesnotexistihope.com"); //to initialize this
variable
          String uri = "http://api.microsofttranslator.com/v2/Http.svc/Translate?text=" + encText +
"&from=" + inputLang + "&to=" + outputLang;
          String authToken = "Bearer" + " " + token;
          try {
                    url = new URL(uri);
         } catch (MalformedURLException e) {
                  e.printStackTrace();
                  log("Unable to resolve URL");
         }
          //connecting to server and getting translations
          HttpURLConnection connectionToServer = (HttpURLConnection) url.openConnection();
          connectionToServer.setRequestProperty("Authorization", authToken);
          String connectionResult=connectionToServer.getResponseMessage();
```

```
            InputStream serverResult=connectionToServer.getInputStream();
            //parsing the output to get the actual translation, the original is an XML
            StringWriter writer = new StringWriter();
            IOUtils.copy(serverResult, writer, "UTF-8");
            String translation = writer.toString();
            String[] resultArray=translation.split(">");
            resultArray=resultArray[2].split("<");
            translation=resultArray[0];
            return(translation);
}
/**
 * Method to read text from file.
 *
 * Calls out method parse for each line to modify it and then method write to write to the output file
 * @param  token to pass on to the parse method
 * @return Void
 */
static void read() throws IOException {
   log("Reading from file: "+fFileName);
   Scanner scanner = new Scanner(new FileInputStream(fFileName), fEncoding);
   StringBuilder text = new StringBuilder();
   String NL = System.getProperty("line.separator");
   try {
          for (int i=0; i<13; i++){
                 text.append(scanner.nextLine()+NL);
          }
          switch (method){
          case 1:
                 token=getToken();
                 log("Applying machine translation from "+inputLang + " to " + outputLang);
                 while (scanner.hasNextLine()){
                        text.append(parse1(scanner.nextLine()));
                 }//while cycle for parsing each line
                 break;
          case 2:
                 log("Applying prefix/suffix method");
                 while (scanner.hasNextLine()){
                 text.append(parse2(scanner.nextLine()));
          }//while cycle for parsing each line
                 break;
          case 3:
                 log("Applying key method");
```

```
                    while (scanner.hasNextLine()){
                    text.append(parse3(scanner.nextLine()));
            }//while cycle for parsing each line
                    break;
     }// end switch
    }//end try
    finally{
     scanner.close();
    }
    write(""+text);
  }
/**
 * Writes text to the file
 *
 * @param  input text to write to the file
 * @return Description text text text.
 */
 static void write(String input) throws IOException  {
              log("Writing to file named " + fFileNameOut + ". Encoding: " + fEncoding);
              Writer out = new OutputStreamWriter(new FileOutputStream(fFileNameOut), fEncoding);
              try {
                out.append(input);
              }
              finally {
                out.close();
                log("File ready");
              }
             }
 /**
  * Gets the translatable text from the input line, translates it using method translate and then
returns the line on it's original form with translated text
  *
  * @param  input line of XML to get translatable text from and return
  * @return a line of XML with translated text
  */
static String parse1(String input) throws IOException{
          String NL = System.getProperty("line.separator");
          Scanner scanner = new Scanner(input);
          if(input.length()<24){
                    return(input+NL);
          }
          else{
```

```
            scanner.useDelimiter(">");
            String Ihalf="";
            String IIhalf="";
                    if ( scanner.hasNext() ){
              Ihalf=scanner.next();
              IIhalf = scanner.next();
              scanner=new Scanner(IIhalf);
              scanner.useDelimiter("<");
              IIhalf = scanner.next();
              IIhalf= translate(IIhalf,token);
             }
             else {
              log("Empty or invalid line. Unable to process.");
             }
                    if (IIhalf==""){
                    return("");
                    }
                    else{
                            return(Ihalf+">"+IIhalf+"</item>"+NL);
                    }
          }
}
 /**
  * Gets the translatable text from the input line, returns the original line with the string replaced by
"X String X"
  *
  * @param  input line of XML to get translatable text from and return
  * @return a line of XML with "X string X"
  */
static String parse2(String input) throws IOException{
          String NL = System.getProperty("line.separator");
          Scanner scanner = new Scanner(input);
          if(input.length()<24){
                    return(input+NL);
          }
          else{
            scanner.useDelimiter(">");
            String Ihalf="";
            String IIhalf="";
                    if ( scanner.hasNext() ){
              Ihalf=scanner.next();
              IIhalf = scanner.next();
```

```
                scanner=new Scanner(IIhalf);
                scanner.useDelimiter("<");
                IIhalf = scanner.next();
                IIhalf="X "+IIhalf+" X";

              }
              else {
                log("Empty or invalid line. Unable to process.");
              }
                    if (IIhalf==""){
                    return("");
                    }
                    else{

                              return(Ihalf+">"+IIhalf+"</item>"+NL);
                    }
          }
}
 /**
  * Gets the translatable text from the input line, returns the original line with the string replaced by
its key
  *
  * @param  input line of XML to get translatable text from and return
  * @return a line of XML with key
  */
static String parse3(String input){
          String NL = System.getProperty("line.separator");
          Scanner scanner = new Scanner(input);
          if(input.length()<24){
                    return(input+NL);
          }
          else{
             scanner.useDelimiter(">");
             String Ihalf="";
             String IIhalf="";
                    if ( scanner.hasNext() ){
                Ihalf=scanner.next();
                IIhalf = scanner.next();
                scanner=new Scanner(IIhalf);
                scanner.useDelimiter("<");
                IIhalf = scanner.next();
                String key="";
```

```
                scanner=new Scanner(input);
                        scanner.useDelimiter("\"");
                        scanner.next();
                        key=scanner.next();
                        IIhalf=key;
            }
            else {
              log("Empty or invalid line. Unable to process.");
            }
                if (IIhalf==""){
                        return("");
                }
                else{

                        return(Ihalf+">"+IIhalf+"</item>"+NL);
                }
        }
}
 /**
  * Writes text to the standard output (command line)
  *
  * @param  writable text
  */
private static void log(String aMessage){
   System.out.println(aMessage);
}
/**
 * Input filename
 */
private static String fFileName="input.xml";
/**
 * Output filename
 */
private static String fFileNameOut="output.xml";
/**
 * Encoding used for writing and reading from files
 */
private final static String fEncoding="UTF-8";
/**
 * Default input language to be translated from
 */
private static String inputLang = "en";
```

```
/**
 * Default output language to be translated to
 */
private static String outputLang = "de";
/**
 * Token used to connect to Microsoft Translator API
 */
private static String token = "";
/**
 * Method selection,  1 = Machine Translation, 2 = Prefix/suffix, 3 = Key
 */
private static int method = 1;
/**
 * Microsof Azure Client ID
 */
private static String clientId = "**************";
/**
 * Microsoft Azure ClientSecret
 */
private static String clientSecret = "****************";

}
```

## Appendix 3. CD with the program code, example Skype XML file and the digital copy of the thesis

Contents of the CD:

1. *input.xml* – an example of the Skype XML file
2. *BA_Thesis_Helen_Kask.pdf* – digital copy of the thesis
3. *Pseudolocalizer.java* – the source code of the application
4. *Readme.txt* – a text file containing instructions how to launch the program
5. *Pseudolocalizer* folder, which contains:
    a. *Pseudolocalizer.jar* – executable jar file, executed from the command line
    *b.* *input.xml*- an example of the Skype XML file, default input of the program.
    c. Various jar packages needed to launch the Pseudolocalizer.jar file.