

# MULTILIZER™

G L O B A L   L O C A L I Z A T I O N



**Getting Started**

---

MULTILIZER™

Getting started with software localization using MULTILIZER

September 1999

Copyright © 1999 Innoview Data Technologies, Ltd. All rights reserved.

MULTILIZER is a trademark of Innoview Data Technologies, Ltd.

Delphi is a registered trademark of Inprise Corporation.

C++Builder is a trademark of Inprise Corporation.

# Program license agreement

---

For MULTILIZER™

- 1. DEFINITIONS.** Software means the enclosed MULTILIZER software program and related documentation. Application Program means a VCL, Visual Basic, WFC, Java or other program designed to accomplish a specific purpose or perform a specific task for the end user. The parties to the agreements (Agreement) are Innoview Data Technologies Oy (Innoview) and the purchaser (you as a Licensee).
- 2. LICENSE.** Innoview agrees to grant to you and you accept a non-exclusive, non-transferable license to use Software to the extent for which a license fee has been paid. You may not sublicense, rent, distribute, lease or otherwise assign your rights in the Software. The license is not a sale of the Software and Innoview retains full title to the software recorded on the media contained in the Software. Innoview reserves all rights not expressly granted to you in this License. Innoview grants to you the right to use one copy of the Software specifically for the purposes described in this Agreement. Except as otherwise expressly approved in writing by Innoview, you may not use the Software at the same time on more computers or computer terminals than the number of authorized copies of this Software that you have purchased.
- 3. TERM.** This License is effective from the date you open this package or use the Software, and shall remain in effect until terminated. You may terminate this License by destroying all complete and partial copies of the Software in your possession. The license shall terminate immediately if you fail to comply with any of its terms, in which case you will certify to Innoview in writing that, to the best of your knowledge, the original and all copies or partial copies of the Software have been destroyed or returned to Innoview.
- 4. RIGHTS IN THE SOFTWARE.** You acknowledge that the Software and any copies of it, regardless of the form or media in which the original or copies may exist, are the sole and exclusive property of Innoview, and you further acknowledge that the Software, including the code, logic and structure, constitute valuable trade secret rights that belong to Innoview. You agree to secure and protect the Software consistent with the maintenance of Innoview's rights in it, as set forth in this License. By accepting this License, you do not become the owner of the Software; Innoview retains full title to the software recorded on the media contained in the Software.
- 5. COPIES AND OTHER RESTRICTIONS.** The Software is protected by copyright law and international treaty provisions. Notwithstanding the copyright, the Software contains trade secrets and confidential information of Innoview. You agree not to disclose or otherwise make available any part of the Software to any third party. You may make copies in machine-readable form of the computer program which is part of the Software, provided that the copies are used only for back-up or archival purposes, that no more than two complete or partial copies exist at any time, and that the copies contain the original copyright notice. You may make unlimited copy of the Application Software and deliver them to the end user with no additional fees or royalties payable to innoview.
- 6. DISCLAIMER OF WARRANTY.** The Software is provided 'as is' without warranty of any kind, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Further, Innoview does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of the Software or its documentation in terms of correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of the software is assumed by you. The above are the only warranties of any kind; no oral or written information or advice given by Innoview or its employees shall create a warranty or in any way increase the scope of this warranty, and you may not rely on any such information or advice.
- 7. LIMITATION OF LIABILITY.** You assume full and sole responsibility for any use you make of the Software and you bear the entire risk of any error in its functioning. You agree that regardless of the cause of any error or the form of any claim, your sole remedy and Innoview's sole obligation shall be governed by this agreement, and in no event shall Innoview's liability exceed the amount you paid for the software alleged to have given rise to the liability. Innoview shall not be liable for any direct, indirect, consequential, or incidental damages (including damages for loss of business profits, business interruption, loss of business information, and the like) arising out of the use or inability to use the software, even if Innoview has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.
- 8. U.S. GOVERNMENT RESTRICTED RIGHTS.** The Software is provided with RESTRICTED RIGHTS. Use duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(2) of the commercial computer Software – Restricted Rights Clause at FAR 52.227-19 and in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFAR 252.227-7013. Contractor/manufacturer is Innoview Data Technologies Oy, Eteläranta 14, 00130 Helsinki, Finland.
- 9. GENERAL.** This license, including the Disclaimer of Warranty and Limited Warranty, merges all prior written and oral communications regarding the Software and sets forth the entire agreement between you and Innoview, unless otherwise stated in a separate written agreement. This License shall be construed in accordance with the internal laws of Finland and all disputes shall have exclusive venue in the state court

in Helsinki, Finland. If any term of this License shall be found invalid, the term shall be modified or omitted to the extent necessary, and the remainder of the License shall continue in full effect.

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU FURTHER AGREE THAT IT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF THE AGREEMENT BETWEEN US WHICH SUPERSEDES ANY PROPOSAL OF PRIOR AGREEMENT, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN US RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

# Table of Contents

---

1	Using this manual.....	1
	Preface.....	1
	Conventions used in this book.....	1
	Typefaces .....	<b>Error! Bookmark not defined.</b>
	Symbols .....	1
	Installation .....	2
	Commercial version .....	2
	Evaluation version .....	2
	Bundle version .....	2
	Getting help .....	3
	On-line help.....	3
	Internet.....	3
	Registration.....	3
	Technical Support.....	3
2	Overview .....	5
	Unique architecture .....	5
	Standard and Professional versions.....	5
	Language support .....	6
3	Basic concepts .....	7
	Internationalization .....	7
	Localization .....	7
	Localized program .....	7
	Single worldwide source.....	8
	Single worldwide binary.....	8
4	Coping with different languages .....	10
	Character sets .....	10
	Single-byte character sets .....	10
	Multi- or double-byte character sets .....	11
	Bi-directional character sets .....	11
	Unicode .....	12
	Singlebyte and double-byte character set trouble .....	12
	Universal character encoding system.....	13
	Text Input .....	13
	Western languages.....	13
	Far Eastern languages .....	13
	Middle Eastern languages .....	13

5	Country specific items .....	15
6	More about localization .....	16
7	How does MULTILIZER work? .....	17
	Dictionary .....	17
	Language Manager.....	17
	Dictionary components .....	18
	Translator .....	19
	Concerns when building a dictionary.....	19
	Coping with words with several meanings.....	19

# 1

## Using this manual

---

### Preface

The purpose of this Getting Started manual is to familiarize you with MULTILIZER™ and the concepts and techniques behind the software localization.

This manual gives a short overview of the world's languages, which is a central issue in software localization. Also, the different scripts are discussed. Later these issues are described in the context of developing software.

After the introduction to software localization related concepts, the MULTILIZER technology is introduced to the reader. This gives an overview of the technical solutions, which make it flexible and scalable even when working with multilingual – or multicultural – software.

This chapter gives important information on the following issues:

- Text style and symbol conventions used in this book
- MULTILIZER Installation notes
- Where to get support.

At the end of this manual, there is a glossary of terms and concepts used in this book.

### Conventions used in this book

#### Typefaces

The following typographical conventions have been used in this book.

Names of windows, menu options and key buttons are printed in **bold Sans serif**.

Texts for figures and references to chapters and sections in this guide are shown in *italic Sans serif*.

Programming language related items are shown in the following manner.

- Names of components, component properties, procedures, and functions are shown in **bold monospaced font**.
- Code listings and URLs are shown in `monospaced font`.

#### Symbols



This symbol indicates that the information given applies to the Standard version only. In front of a header it applies to the whole chapter, otherwise it applies to the current paragraph.



This symbol indicates that the information given applies to the Professional version only. In the front of a header it applies to the whole chapter, otherwise it applies to the current paragraph.



**MORE INFO**

The More info symbol is used when there is additional information available either in the appendices of this document, in MULTILIZER online help or on MULTILIZER web-pages at: <http://www.multilizer.com>



**NOTE!**

The note symbol is used in order to give emphasis for certain tasks or issues which are of big importance in the current topic.



**TIP!**

The text marked with the Tip symbol gives useful hints, which may simplify tasks described in the current chapter.



**WARNING!**

The Warning symbol is used whenever there may be a possibility to lose data or experience other kinds of damage. The normal context for this symbol indicates that you may lose your translation data if you proceed.



**CHAR.SET**

This symbol is used in issues describing different character sets. Character sets are one central issue to be taken in consideration when localizing software.

## Installation

### Commercial version

MULTILIZER is installed from the CD-ROM. Refer to the instructions in it for installing the software.



**NOTE!**

If you have a previously installed an evaluation version on your computer, remember to uninstall it before installing the commercial version.



**TIP!**

For commercial version users, there are available free minor updates on the MULTILIZER support pages:

<http://www.multilizer.com/support>

### Evaluation version

Evaluation versions of the software are available at the following MULTILIZER sites:

<http://www.multilizer.com> (Europe)  
<http://usa.multilizer.com> (North-America)

### Limited version

A limited version of MULTILIZER is included with the following software development tools:

- JBuilder 3
- C++Builder 4

The limited version does not have all the features of MULTILIZER.

## All versions



After you have installed the software, you have to add the MULTILIZER components to the IDE of your compiler. In addition, the online help may be linked to the help system. See more on the online help topic 'Getting started'.

## Getting help

### On-line help

This manual is a getting started manual. It does not contain any reference information about MULTILIZER. For complete instructions on using MULTILIZER, see the online help.

### Internet

You can also get more information and tips on our Internet sites.

<http://www.multilizer.com> (Europe)  
<http://usa.multilizer.com> (North-America)

Currently all the WWW files are located on those two servers. In the future we will replicate the sites to Australia, the Middle East and Asia. You can choose the mirror on our front page:

<http://www.multilizer.com/index.html>

### Registration



By registering the product you will get technical support as defined in the Software License Agreement.

Although the MULTILIZER package contains the registration card, we strongly recommend that you register the software at our web site:

<http://www.multilizer.com/support/register.htm>

This will enable your personal account in our Internet server. Using the account you can download new versions and bug fixes. You can also join a mailing list that keeps you updated about Innoview MULTILIZER and multilingual globalization technology.

You can download the newest MULTILIZER version at (after you have registered your software at our Internet site)

<http://www.multilizer.com/support>

### Technical Support

**Support Center** If you have a question about MULTILIZER, look first at online help. If you can not find the answer at the online help check it at the support center:

<http://www.multilizer.com/support/index.htm>

**Support News** Also consult the MULTILIZER newsgroups. (NNTP Server [multilizer.com](http://www.multilizer.com)).

Discussion group	Topics
<code>multilizer.language-manager</code>	Language Manager
<code>multilizer.dictionary-server</code>	Dictionary Server
<code>multilizer.delphi</code>	Delphi
<code>multilizer.cbuilder</code>	C++Builder
<code>multilizer.visual-basic</code>	Visual Basic
<code>multilizer.java</code>	Java

**Sending your program source to technical support**

Sometimes our technical support person will ask you to send a sample application. Try to create a simple program that demonstrates your case. Try not to use any 3<sup>rd</sup> party components. If they are needed to demonstrate the problem, include the 3<sup>rd</sup> party components in the package, or give the instructions how to download the. ZIP source code files and the dictionary files (\*.LMP, \*.LMM, and \*.MLD or \*.TXT) into a single zipped file and email it to our technical support or to the news group.

Please include the version number of your MULTILIZER, compiler version used and the operation system (version/language). This will greatly speed up the solving of your problem.

Any source codes we obtain through bug reports are handled strictly confidentially.

# 2

## Overview

MULTILIZER provides an easy and flexible way to localize your applications. MULTILIZER goes beyond localization – it makes your programs multilingual. A multilingual application supports multiple languages and the user can switch the language on the fly.

### Unique architecture

MULTILIZER uses unique *Dictionary-Translator Architecture* where the layout of the form and the translation table is kept separate. This makes it possible to support multiple languages with one resource. It is also possible to reshape forms without breaking the translation in any way, and add new languages without using development tools.

For MULTILIZER users, this architecture makes possible a very practical approach to localization:

- Human language issues are conducted with the MULTILIZER Language Manager utility.
- Programming issues are done by using the MULTILIZER components.

These are both included in every MULTILIZER package.

*Language Manager* provides timesaving features for working with human languages: Wizard technology, re-use of translations, built-in quality assurance indicators, glossary-based pre-translation technology, and enterprise-wide leveraging capabilities.

The *components* provide the developers with an easy-to-use yet powerful interface for localization programming. Developing the RAD way, localized software's behavior is controlled through the component properties, events and methods.

### Standard and Professional versions



There are two different versions of MULTILIZER: *standard* and *professional*. The difference is in the support for character sets:

- The Standard version supports Western character sets (Latin, Greek, Cyrillic)
- The Professional version supports Western, Middle Eastern and Far Eastern character sets.

This manual covers both versions. Whenever there is a difference in the features depending on the version, it is marked either with the Pro symbol or with the Std symbol. To see the symbols, cf. *Previous chapter*.

Both MULTILIZER versions work using the same technology – the difference is in the supported languages, which will be further explained in the next paragraph.

## Language support

The standard version supports all languages based on European character sets. These are the:

- Latin character set (e.g. English, French, German, Spanish, Swedish, Finnish, Turkish).
- Cyrillic character set, e.g., Russian and Ukrainian.
- Greek character set, e.g. Greek.



The professional version adds support for all languages based on non-European character sets. In addition to the European character sets, the Professional version supports:

- Multibyte languages (Chinese, Japanese, Korean).
- Bi-directional languages (Arabic, Hebrew and Farsi). Bi-directional languages are generally written from right to left, numbers are written from left to right. This is not supported in the 16-bit Windows version.

With MULTILIZER Professional you can create *single worldwide binary*. The standard version makes it possible to create *single European wide binary*.

Ultimately, the support for languages depends of the OS support for language specific features.

# 3

## Basic concepts

This chapter explains some key issues about software internationalization, localization and globalization. For terms used in this chapter, cf. Appendix A: Glossary, p. 22.

### Internationalization

In software internationalization the hard-coded language or country dependent information is removed. In practice this means that no language specific information, currencies, dates, times etc. should be inside the program code.

Software internationalization is the first phase which has to be done in order to make the software adapt to the target country. Depending of the type of software and how it has been programmed, this phase may be very time-consuming.

The programmer typically does this.

### Localization

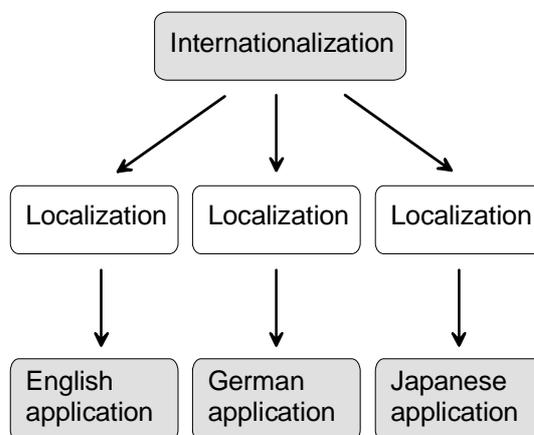
Localization means converting a program to a local market. The simplest way is to translate all the strings into the local language. In most cases you also have to make slight modifications to the program to meet the local standards and culture.

In addition the software has to be prepared to support the target country's character set and language.

The linguist typically does this.

### Traditional Globalization

The following picture demonstrates the traditional globalization process. It starts with internationalization. The process continues with the localization where the linguists translate the user interface, manuals, etc. The result is several localized applications.



*The traditional globalization process*

Each localized program can only support one language.

For example the native language might be English. If you decide to localize the program to German, French, and Spanish you will have four different versions of the program - each supporting one language.

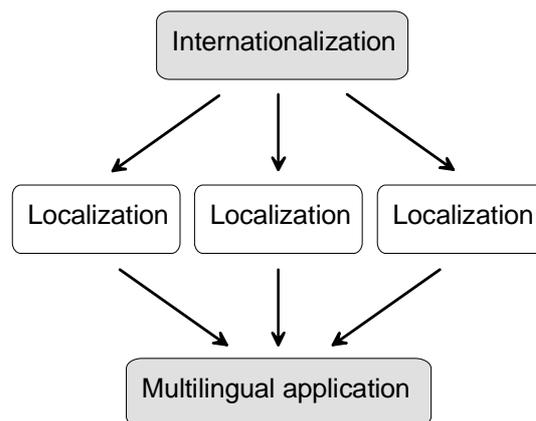
## Single worldwide source

Single worldwide source means that you have only one version of your source code. You either compile or include new resources for each localized version.

Traditional globalization uses single worldwide sources and localization. The methods for doing the internationalization and localization may vary a lot. The time spent on these phases may become the most decisive factor in software delivery dates, as well. Traditional localization techniques may take several months to complete and meanwhile the source should be frozen in the current version.

## Multilingual Globalization

A multilingual application takes localization one step further, in that a single-file application supports multiple languages, including the capability of switching languages at run-time. Having only one set of source and resource files to maintain obviously makes the development process more cost-effective.



*The multilingual globalization process.*

MULTILIZER™ is a tool for all three parts of the process. The MULTILIZER documentation and examples help you to internationalize your applications. The Language Manager utility helps the linguist localize the application. Finally, the MULTILIZER component helps the programmer build a multilingual application.

Compared to the traditional techniques, MULTILIZER's *Dictionary-Translator Architecture*, Language Manager and the components make it possible to save a lot of time in the globalization process. If MULTILIZER is used from the beginning of software development, country specific items are never hard-coded. This feature saves you a lot of time.

Due to the architecture, you don't have to internationalize the program source code in the traditional way - i.e. isolating all strings to the resource. You can go through the whole localization process with a minimum impact on the source code.

With MULTILIZER, you can continue developing the software even when localization takes place.

## Single worldwide binary

Single worldwide binary means that there is only one version of your program. This version can handle all of the world's languages. MULTILIZER makes it possible to create a single worldwide binary.

# 4

## Coping with different languages

---

One main task in software localization is to make the software work in the language and character set of the target country.

Originally, computers were designed to work with character sets including only those characters needed in English. When computers became commercially available everywhere in the world, there arose a need to include the target country's characters into the character set as well.

To be able to handle a specific language in the computer, it must be possible to handle its character set. There must be methods for inputting, storing and outputting characters. Thus, localizing applications involves

- processing of character sets and
- accommodation of the application's I/O methods for the current language.

The following chapters introduce the different types of scripts. Later, the most important of these will be discussed in the context of today's information technology.

### Character sets

All languages in the world can be divided into three basic groups, depending on the type of character set they use. The groups are:

- Single byte character sets.
- Multi or double byte character sets.
- Bi-directional character sets.

This division is rather a technical one, but as a matter of fact, it reflects three major cultures as well. The groups mentioned above could be rewritten in the following way:

- European character sets.
- Far Eastern character sets.
- Middle Eastern character sets.

### Single byte character sets

Single byte character sets (SBCS), sometimes called single byte left to right, contain a maximum of 256 characters. Each character is stored in one byte. The text is written from left to right. Latin (e.g. English, German, French, Spanish, Finnish, Swedish), Greek and Cyrillic (e.g. Russian, Ukrainian) character sets are single byte.

These character sets evolved in Europe, and they started from the old Hellenistic culture. It is very possible that these scripts came into Europe from Mesopotamia via the Near East, though.

## Greek

The ancient Greek character set was derived from Linear B, which was similar in structure to Japanese. Later, due to the Dorian invasions, a script based on the North Semitic model was adapted. As in modern Middle Eastern scripts, this was written from right to left.

Later, such inventions as the vowels a, e, i, o, u made it the most successful and the most practically useful of the world's scripts at that time.

Modern Greek script has undergone only a few changes since the classical Greek period. Most of the changes are phonological.

## Latin

Old Greek script expanded over Southern Italy and came into Roman hands. The alphabet was simplified. Through the influence of the Roman Empire, the Latin alphabet came into use in the whole Western civilization. On the basis of Latin script, the modern European character sets developed.

Thus, these alphabets are very denominative for European culture(s) and for those countries where the Europeans established their colonies. The strong cultural expansion from the XV<sup>th</sup> Century on has exported – especially the Latin alphabet – all over the world. In South and North America, Africa and Australia, the Latin alphabet is almost the only one in use.

All of these character sets have implemented accent marks to denote special sounds, i.e., the characters are based on a base character and the phonetic difference is marked with an accent. In addition, some additional characters have been taken in use.

For example, in German, 'ß' is used to denote 'ss' when it is in a certain place. In addition, in African languages characters like |, ||, ‡, ≠ are used to mark non-pulmonic sounds (clicks) not used in European languages.

From the Middle Ages some ligatures survived: French œ, Danish æ. The best known is the ampersand '&', a ligature of the Latin word 'et' (*and*).

In addition, languages using different character sets and scripts are often transliterated in literature and schoolbooks into the Latin character set. For transliteration purposes, additional diacritics are attached to base characters.

## Multi or double byte character sets

Multi byte character sets (MBCS), sometimes called double byte (DBCS) or Far East, contain more than 256 characters or idioms. Each character is stored either in one byte or two bytes. The text is written mainly from left to right top to bottom but sometimes from top to bottom left to right. Chinese, Japanese and Korean character sets are multi byte.



Only the professional version of MULTILIZER supports multi byte languages.

## Bi-directional character sets

The bi-directional character set (BiDi), sometimes called single byte bi-directional, contains a maximum of 256 characters. Each character is stored in one byte. The text is written mainly from right to left but sometimes from left to right. Arabic and Hebrew character sets are bi-directional.

The ancient Arabic script was a derivative of the Nabataean consonantal script, which was used two thousand years ago. Later, it was influenced by Mesopotamian Kufic

scripts. From the eleventh century onwards, a flowing cursive style was developed, and it became the Arabic script commonly used. This script underlies most contemporary type-fonts.

Arabic script is used for a number of important languages: Persian, Urdu, Pashto, Baluchi, Kurdish, Lahnda, Kashmiri, Sindhi and Uighur.

Since the phonological inventories between these languages may differ a lot from those of Arabic, the script has had to be augmented and adapted to meet the new demands made upon it. In some languages, such as Sindhi, certain Arabic letters are adapted to denote multiple sounds. On the other hand, in some languages there are a lot of redundant letters – in Persian there are four Arabic letters pronounced exactly in the same way.

Arabic script has been used in many other languages. However, many of them have abandoned Arabic script for Latin. Among them there are languages like Indonesian (Malay), Hausa, Somali, Sudanese, Swahili and Turkish. Partially this is explained by the last 400 years' aggressive expansion of the European cultures. In addition Turks wanted to modernize the country at the beginning of the 20<sup>th</sup> century and they adopted the Latin alphabet. Several Caucasian languages, e.g. Chechen, Kabardian, Lak, Avar, Lezgi used the Latin alphabet for a while. Due to the expansion of Russian power, the Cyrillic alphabet was implemented.

Arabic script is used nowadays in a widespread area from the Atlantic coast in Morocco to India. It is very probable that these areas remain as users of this script. This area has been politically quite unstable, but economical growth is expected, thus bringing information technology to these countries. This will open a need for supporting Arabic script in software.



The professional Java and Windows (32-bit) versions of MULTILIZER support bi-directional languages.

## Unicode

### Single-byte and double-byte character set trouble

Single-byte character sets are able to support a maximum of 256 different characters. Originally this was sufficient to cover English and some other European languages.

However, there are many languages using Latin characters with diacritics. Because there was no place for them, different code pages were implemented. Therefore, e.g., Romanian uses a different code page than French. To make the software run in both languages, the code page had to be changed, which often meant rebooting computer or installing a specific OS.

For Far Eastern languages Double-byte character sets were implemented. The scripts of these languages contained more than 256 ideographs. Thus the characters were encoded in two bytes.

These single or double-byte character sets had the problem that one code (such as Hex 67) could hold different characters, according to the code page used. Therefore Unicode was introduced.

### Universal character encoding system

Unicode is a fixed-width, 16-bit worldwide character encoding system developed by the Unicode Consortium and endorsed by Window NT. It is a "universal" character set of approximately 35,000 characters encompassing all major character set standards.

In Unicode, each character has a unique code.

The Unicode Consortium, a nonprofit computer industry organization, continues to maintain and promote the system.

You can get more info on Unicode at: <http://www.unicode.org>



## Text Input

### Western languages

Western alphabets are based on characters. There are normally a very restricted number of characters with which all the words in a western language can be formed. Therefore most characters can be input with one character stroke on the keyboard.

Different Western languages use keyboard layouts that differ slightly from each other. This is due to the input of some language specific characters (Cf. *previous chapter*).

MULTILIZER Language Manager is able to change the keyboard layout to match the current language being edited.

In addition Language Manager helps in entering Latin characters with diacritics: you can use existing characters or define your own compose character sequences for entering accented characters. This makes the working easier, e.g., with a US keyboard.

### Far Eastern languages

Far Eastern languages need a specific way for inputting ideographs. There might be thousands of characters which should be input. For that reason, Far Eastern Windows language editions ship with Input Method Editors (IME).

Using the IME, users can compose each character in one of several ways: by radical, by phonetic representation or by the character's numeric code-page index.

MULTILIZER Language Manager uses the IME provided by Far Eastern Windows language editions for inputting those languages.

### Middle Eastern languages

Middle Eastern languages have a restricted number of characters in their alphabets, like Western languages. The problem of inputting characters in these languages lies in the following reasons.

- text is written from right to left, numbers from left to right
- Arabic characters obtain different forms depending on where they are located in the word (initial character, in the middle of the word, final character or isolated character).

Arabic or Hebrew language editions of Windows are needed for inputting Middle Eastern characters in Language Manager.

For more information on editing a specific language in Language Manager, cf. Language Manager documentation.



# 5

## Country specific items

---

Besides making the software work in different languages, the software must also be fine-tuned to work with country specific standards. Country specific standards are later referred as locale data.

In fact locale information tells what language is spoken in the specified locale. Therefore the language can be considered as a subset of the locale.



In Language Manager a specific locale is referred to using the language and the country's name in parenthesis. Thus, selecting a locale like e.g., French (Canada), means that the country is Canada and the language is French.

# 6

## More about localization

---

The following list contains various sources on where to get more information about localization:

- Developing International Software for Windows 95 and Windows NT, Nadine Kano, Microsoft Press, 1995. This is an excellent book about developing international software.

You can find this book in MSDN, too.

- WIN32 API documentation
- <http://www.xerox-emea.com/globaldesign/>

# 7

## How does MULTILIZER work?

---

MULTILIZER is based on a translation-dictionary mechanism. There is a Dictionary containing the translations and a translator, which translates the program strings on the fly.

MULTILIZER dictionary-based translation keeps the language maintenance separate from programming, which makes big projects easier to manage. In addition, the developer can make the decision where to store the translation data used by the software. He can choose between standard text files, Unicode, database tables, a dictionary server, binary file etc.

The Dictionary is edited and maintained with Language Manager. The Dictionary is implemented in the programming phase as a Dictionary component. The translator component does the actual translation. For any multilingual program you develop, you have to be familiar with:

- Language Manager
- Dictionary component(s)
- Translator component(s)
- Module component(s)

The following chapter describes in brief the issues mentioned above.

**MULTILIZER does not translate your words.**

It is important to understand that MULTILIZER is not a tool that automatically translates words or phrases into another language. Instead it provides a mechanism that uses precompiled translator tables, dictionaries, to make the program multilingual. It is your (or your translator's) job to translate the strings in your program. Language Manager makes this translation as easy as possible.

### Dictionary

The underlying concept behind MULTILIZER is to state each child component of a form, and translate its string type properties into another language.

The translation is done by finding the corresponding string from the lookup table and replacing that string with the new one. This architecture is called *MULTILIZER Dictionary-Translator Architecture™* (MDTA™).

### Language Manager

The lookup table mentioned above is called the translation table, and contains strings for each language that the application uses. In addition, the dictionary contains a language table and a locale table.

Language Manager is used for creating and maintaining all the tables in the dictionary. It is a Windows utility, which makes the dictionary maintenance as convenient as possible. In addition you are able to maintain all the locale-specific information apart from your

software. This makes the development work clearer and the work can be shared among several persons.



For additional information on Language Manager, cf. **langman.pdf**.

### Language table

In addition to the translation table, MULTILIZER (and the Operating System) requires information about the current language. To provide this information, each dictionary contains another table called the language table, which contains information about the languages in the dictionary.

For every column in the translation table there must be a row in the language table (in the same order). The first row of the language table is the first column of the translation table, and so on. The language table contains the native name of the language and the name in English. The table also contains the language IDs, and the default font of the language.

### Locale table

Simply switching between languages is not always the only requirement of a multilingual program. The program may have to adapt to some local customs and standards such as date and time formats, currency and the measurement system.

To provide this facility, MULTILIZER uses locale data. Depending on the OS, MULTILIZER uses either the locale OS locale information or a third table called the locale table. The locale table is generated with Language Manager's *Dictionary Wizard*.

With MULTILIZER you ensure that your localized program follows the locale support provided by the OS. This ensures maximal compatibility. In addition, you are able to specify locales not supported by the OS, thus giving you the ability to localize your software for countries/locales not supported by the OS.

Depending on the target countries and your software platform, you may or may not need the locale table.

### Dictionary components

Once you have created a dictionary with Language Manager, you have to attach the information in it into the software. This is done by using a dictionary component.

There are different kinds of dictionary components, each implementing the data in the dictionary in a different way into the software. By using an appropriate dictionary component, you can implement the dictionary as an external text file, a binary file, a database table, etc.

Different implementations of the dictionary give you the freedom to choose the best possible method for your project. Thus, you are able to optimize your localized program according to the characteristics of your project.



Generally the Dictionary component is placed on the application main form. Thus, you need one Dictionary component per application.



For comprehensive information of the dictionary component use, cf. MULTILIZER online help.

## Translator

The translator is the part of the MULTILIZER architecture which does the translations. It uses the dictionary for making the application either multilingual or for adding support for a specific locale.

The translator is implemented as a component in MULTILIZER. It states each component of the host form, and translates the string type properties into another language. Setting the Target properties you can specify which strings to translate in your application.



Translator components are placed on every form where you want to enable automatic form translations. By inheriting a form with a Translator component, you just need to add the component once.

The translation is done by finding the corresponding string from the lookup table and replacing that string with the new one. This is done by assigning a Dictionary component to the Translator component.

Together the Dictionary components and Translator components work as the powerful core for your multilingual or localized applications. Using them, you easily get started with localization. The architecture provides scalability and flexibility for software development.

## Concerns in building a dictionary

As mentioned in the previous chapter, the translator components translate the targets on run time. The language used in software development is called the native language. When running the localized/multilingual program in English, the translator components translate from Native to English. When run in French, Native is translated into French.

For a software developer, using English as a native language, at first it sounds strange that MULTILIZER translates from English 'Native' to English. However, in order to make the architecture and functionality of MULTILIZER as simple as possible, the same translation is applied to every target language.

Furthermore, in localizing software there are certain issues which make the use of 'Native' language very convenient. The next chapter explains one reason for using 'Native' language.

### Coping with words with several meanings

A problem arises when the same native word is used to convey two different meanings, depending on the context in which it is used. For example, the word 'firm' can mean a 'company', or 'hard'. In other languages, separate words may be used to convey the two meanings, but because the dictionary index key must be unique it is not possible to specify the same word twice.

English	Finnish	Swedish
firm	yritys	Företag
firm	luja	Stark

This dictionary model is called the *flat* dictionary. The first column creates the key.

### Using context sensitive dictionaries

This would be a problem because the translator component would not know which "firm" to use. To solve this the dictionary can use the string context. This means that every

string has a context where it belongs. For example the “&Browse...” string belongs to the caption property of the browser button. Without the string context it would be possible to have two equal native strings that have different translations. The following dictionary has two “firm” strings having different contexts.

Native	Form	Component	English	Finnish	Swedish
...					
firm	Main	Label1	firm	yritys	företag
firm	Main	Label2	firm	luja	stark
...					

This dictionary model is called the *context sensitive* dictionary. The first three columns create the key. Unfortunately, context sensitive dictionaries are not available in every platform. The following table contains the supported dictionary formats of the MULTILIZER platforms. The default format is printed in bold.

Platform	Flat	Context sensitive
VCL	yes	<b>yes</b>
Visual Basic	yes	<b>yes</b>
Java	yes	<b>yes</b>

There are two solutions for ambiguous words without using context sensitive dictionaries. Either use a ‘native meta’ language or insert a space character after the word.

#### Using a ‘meta’ language as the native language

With this technique, the native language is a kind of ‘meta language’. For example it can be English whenever an unambiguous word is used but in the case of an ambiguous word the English word must be altered slightly to make it unique.

Native	English	Finnish	Swedish
...			
firm	firm	yritys	företag
firm1	firm	luja	stark
...			

The first ‘firm’ is unchanged, but the second ‘firm’ is actually ‘firm1’. Use ‘firm1’ in the source of the application whenever you want to refer to firm meaning hard, solid, etc.

We recommend the use of this technique whenever the platform does not support context sensitive dictionaries.

#### Using space characters

Another solution is to add space character(s) to the native word or sentence. This extra space makes the distinction between the two meanings of the word.

English	Finnish	Swedish
...		
“firm“	yritys	företag
“firm “	luja	stark
...		

The later ‘firm’ contains one space as the last character.

We do not recommend this technique except in cases where the size of the dictionary is explicitly the key issue.



Language Manager makes it easy to find multiple appearances of Native language strings in the source code: pointing on the flag bar on the left side you can see map info, i.e., the information on where the strings are located in the source code. This helps in finding the ambiguous words in the program source.

### **Native language concerns**

Native language is the key field in the translation table. Whenever an item is translated in the program, a search is performed in the translation table. The search mechanism is optimized for maximum speed, which disallows certain characters in the Native language.

---

## Appendix A: Glossary

---

This glossary describes common terminology used in software localization and in MULTILIZER.

### Code pages

A code page is a code array that maps the integer code to the character of the character set. The first 128 items of every code page contains the ASCII characters. The remaining items depend on the character set.

*Windows 3.1* supports only one code page. It is the code page of the system and it is bound on the language version of your operating system.

*Windows 95* and *Windows 98* support multiple code pages but only one can be the system code page. It is bound on the language version of your operation system. It can not be changed.

*Windows NT* and *Windows 2000* support multiple code pages but only one is active at a time. The system must be rebooted after a code page change.

*Java* uses Unicode so there is no need to use code pages.

### Dictionary

MULTILIZER uses dictionaries to store the translation data. Each MULTILIZER edition contains one or more dictionary components that access that data. In most cases all the translation data of your application is stored in one dictionary.

### Globalization

Globalization is the compound of tools and methods which are applied to software in order to make it work globally. Thus, the globalized software works with the appropriate features of each of the target countries.

Globalization can be seen as the sum of internationalization and localization: when globalizing the software, it first has to be internationalized and after that localized. Properly developed software targeted for many locales must go through both internationalization and localization.

### Internationalization

Internationalization is the first phase in making software for global markets. Software internationalization means the act of removing country and locale specific features hardcoded in the software's source code.

However, internationalization tends to be used in the meaning of globalizing software.

### Language IDs

**Language ID** The language ID specifies the language (e.g. English, Finnish, Japanese, etc.).

MULTILIZER uses Windows's primary language IDs (e.g. LANG\_ENGLISH), and Java's language codes as defined by ISO-639 (e.g. "en").

**Country ID** The country ID specifies the country of the language (e.g. English in United States, English in Great Britain, English in Canada, etc.).

MULTILIZER uses Windows's sub-language IDs (SUBLANG\_ENGLISH\_US), and Java's country code as defined by ISO-3166 (e.g. "US")

## **Linguist**

The linguist is the human translator who translates the dictionary into the target language(s).

## **Locale**

A locale is a combination of language and country IDs. In other word it is a language spoken in a country (e.g. German spoken in Austria). There can be multiple locales for one language. For example, English (United Kingdom), English (United States), English (Canada), etc.

## **Localization**

Localization is the act of applying country specific features to the program. The name is derivative of locale, which is an OS-specified set of items of the target country's denominative features.

Software localization aims to make the software reflect the target country's cultural features, in order to make the software customer-friendly.

Some countries need several localized versions of the software. For instance in Canada, there is a need to produce both English and French locale versions of the software.

## **Module**

Module is a MULTILIZER component that translates the complex properties of one ore more 3<sup>rd</sup> party component.

## **Translator**

Translator is a MULTILIZER component that translates the form (window) from the native (original) language to the active language just before the form becomes visible. Translator uses the translation data provided by the Dictionary component.