

Hands-on: L10N Development and Porting

Martin Holmichel – Project Lead Porting

Nils Fuhrmann – Co Lead I18n and L10N

Janos Noll

- Prerequisites for a port

- Hardware

- 64 MB Ram

- Graphical User Interface

- Operating System

- Memory Model that allows memory allocation \geq 4 MB

- Shared Libraries (needed for UNO model)

- Processes

- Threads

- Signals, Pipes

- Graphical User Interface with \geq 256 colors

- FileSystem

- Prerequisites for a port

Software

C++ Compiler with (working) support for

- Templates
- Exceptions
- gcc3 recommended

Standard Template Library (STL)

- STLport recommended
 - . eh_test (exception handling process should be passed)

Shell

- Bash or csh compatible

Perl

- Perl 5.6 or better recommended

JDK 1.3 or higher recommended

- Build without JDK is possible, but difficult

- Set up the Build Environment

- Adapt configure.in script

- Define Environment for new platform, e.g.

- GUI UNX

- CPU

- Toolchain

- Setup specific makefile in solenv/inc

- Name of compiler, linker

- Define switches

- Run configure

Hands-On L10N and Porting

- First steps: compile

STLport
dmake
soltools

- Porting

sal/osl (Operating System Layer)

Implementation of Interfaces to OS

- Processes, Thread, FileIO, etc.

vcl

Provides Layer for for

- X11 Window System
- MS Windows
- MacOSX (aqua)

Hands-On L10N and Porting

- Uno
 - Cpputools ?
 - Bridges
- Setup / Scp / readlicense_oo
 - Installation
- Sysui
 - Systemintegration (KDE, CDE, Gnome)
- Compile the rest and test

- Testing the OpenOffice.org port

Unit tests:

Are the tools (dmake, makedepn running)

sal/test

vcl/workben

cppu/test

Create installation set (instsetoo)

Full tests:

Smoke test

API Testing (qadevOOo)

Resource and update Test (qatesttool)

Hands-On L10N and Porting

- Prerequisites for starting localization of OpenOffice.org

A running OpenOffice.org build environment with possibility to rebuild OpenOffice.org sources

Platform of your choice as supported by OpenOffice.org

Downloaded sources including all OpenOffice.org modules

Downloaded solver including all build output mandatory for creation of installation sets

Possibility to translate UTF-8 encoded files

- Two solutions for driving localization of OpenOffice.org

Native language support

- Unique identification of language via numeric/symbolic/ISO code

- Supports building different languages at once

- Restricted to fixed set of languages

- Source code has to be adopted for new languages

L10N-Framework

- Common solution for new languages

- One representation for all languages (99/extern)

- Restricted to one language per build

- No source code adoption for new languages

- Reflection of translations nested inside the code to cvs not possible

- **Process of translation**

- Extraction of strings and messages**

- Execution of “localize” inside the build environment leads to gsi file which includes strings and messages in source and target language (works on native lang. and framework)

- Translation of strings and messages**

- Currently no translation infrastructure provided by OpenOffice.org

- Merge of translated strings into the code**

- Execution of “localize” inside the build environment reflects the strings back to the code (works on native lang. and framework)

- Rebuild of installation sets**

- How to process new and changed strings

Problem: Strings are in change, new Strings are added to OpenOffice.org

Solution: Comparison of strings in source language nested in old and new gsi file (recommended: English or German as source)

Source string changed

- Target string has to be reworked

Source string new

- Target string has to be translated

Source string equals

- Target string OK

Potential solution:

Generation of incremental gsi files

Import of gsi files into database and status handling during import

- What's coming

- New xml based file format for documentation/online help

- Integration into common l10n processes

- Representation of languages by ISO codes instead of numeric/symbolic representation

- Leads to migration of L10N-Framework to native language support

- Translations nested inside source file dependent gsi files

- Keeps the pure source code small

- Leads to “pre-compilation” during build process to reflect translation to the code temporarily