openCRX Language Localization Guide

Version 1.5.0

www.opencrx.org

openCRX Language Localization Guide: Version 1.5.0
by www.opencrx.org

Table of Contents

List of Figures

List of Examples

Chapter 1. About this Book

This book describes which *openCRX* files are language-specific and how you can localize *openCRX* for a (new) language or a set of (new) languages.

Who this book is for

The intended audience are *openCRX* administrators.

What do you need to understand this book

This book describes how to get *openCRX* to work with different language files and how to create (new) language files for *openCRX*. You should be comfortable with editing XML files.

Chapter 2. Prerequisites

This book assumes that you have downloaded an *openCRX* core distribution and **set up *openCRX* for Ant** as described in the *openCRX* README (including the installation of openMDX).

Creating/changing language-specific files does not require a full installation of *openCRX*, although it is helpful to have access to a working installation in order to test your changes. You can create/edit language-specific files without installing an application server or a database.

Chapter 3. Language-specific files

All the language-specific files of *openCRX* are contained in the file **opencrx-core-CRX.war**, which is included in the file *opencrx-core-CRX-web.ear* included in the *openCRX* distribution (please note that this file may already be exploded/unzipped in which case you will have a *directory* called **opencrx-core-CRX.war** instead of a file). Language-specific files can be grouped as follows:

- **user interface configuration** (XML files containing language-specific labels and tool tips) – note that the directory **…opencrx-core-CRX.war/config/ui** contains subdirectories of the form *xx_YY* for each implemented locale; these subdirectories contain the language-specific XML files – the files for US English, for example, are located in the directory …**opencrx-core-CRX.war/config/ui/en_US**
- **code tables** (XML files containing the mapping of codes to the respective language-specific texts) – note that the directory **…opencrx-core-CRX.war/config/code** contains subdirectories of the form *xx_YY* for each implemented locale; these subdirectories contain the language-specific XML files – the files for US English, for example, are located in the directory …**opencrx-core-CRX.war/config/code/en_US**
- **strings used by JSP** – note that the directory **…opencrx-core-CRX.war/config/texts** contains subdirectories of the form *xx_YY* for each implemented locale; these subdirectories contain language-specific text files named *opencrx.text.properties* and *text.properties* – the file for US English, for example, is located in the directory …**opencrx-core-CRX.war/config/texts/en_US**
- **html pages** (e.g. help pages like helpSearch_xx_YY.html where *xx_YY* reflects the locale, e.g. *en_US*)
- login page **Login.jsp** – the login page is a special case and if you want to offer your users a login page in a language that is not configured by default you will have to modify the file **…opencrx-core-CRX.war/Login.jsp**.
- **language configuration** – the XML file **web.xml** contains the language configuration of *openCRX*

Note: openCRX locale IDs are based on a widely accepted standard where the locale ID **xx_YY** is composed of an *alpha-2 language code* **xx** (see *http://en.wikipedia.org/wiki/ISO_639*) and an *alpha-2 country code* **YY** (see *http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2*).

The following directory listing gives an overview of the structure discussed above.

Example 3-1. language-specific files in opencrx-core-CRX.war

```
opencrx-core-CRX.war
|  ...
|  helpJsCookie_de_CH.html
|  helpJsCookie_en_US.html
|  helpSearch_de_CH.html
|  helpSearch_en_US.html
|  ...
\--WEB-INF
   |  ...
   |  web.xml
   |  ...
   +--config
   |  +--code
   |  |  +--de_CH
   |  |  |     accountcategory.xml
   |  |  |     ...
   |  |  |     utcoffset.xml
   |  |  |
   |  |  \--en_US
   |  |        accountcategory.xml
   |  |  |     ...
   |  |        utcoffset.xml
   |  |
   |  +--texts
   |  |  +--de_CH
   |  |  |     opencrx.texts.properties
   |  |  |     texts.properties
   |  |  |
   |  |  \--en_US
   |  |  |     opencrx.texts.properties
   |  |        texts.properties
   |  |
   |  \--ui
   |     +--de_CH
   |     |     abstractcontract.xml
   |  |  |     ...
   |     |     wf.xml
   |     |
   |     \--en_US
   |           abstractcontract.xml
   |  |  |     ...
   |           wf.xml
   ...
```

Important In the Open Source distribution of *openCRX* the user interface configuration files for *en_US* contain the complete customization information in addition to the language-specific text strings.

Warning Modifying language-specific files requires an editor capable of handling files in **UTF-8** format. The UTF-8 encoding of an XML file is indicated at the beginning of the file with <?xml version="1.0" encoding="UTF-8"?>. More information about encoding of XML files is available at *http://www.w3schools.com/xml/xml_encoding.asp*. If you modify language-specific files with an editor that cannot handle UTF-8 encoded files properly you risk running into problems when starting *openCRX* with such modified files because the XML importer will not be able to correctly import them.

Chapter 4. Create files to support a new locale

The following steps are required to create new language-specific files to support a new locale *xx_YY*:

1. expand/unzip the file **opencrx-core-CRX-web.ear** (on the Windows platform you can use Winzip, *7zip (http://sourceforge.net/projects/sevenzip/)*, etc.)
2. expand/unzip the file **opencrx-core-CRX.war** to a temporary working directory get access to all the language-specific files (on the Windows platform you can use Winzip, *7zip (http://sourceforge.net/projects/sevenzip/)*, etc.)
3. create *user interface configuration files* for the new locale *xx_YY* and translate the relevant text strings
4. create *code table files* for the new locale *xx_YY* and translate the relevant text strings
5. create *texts.properties files* for the new locale *xx_YY* and translate the relevant text strings
6. create *html help pages* for the new locale *xx_YY* and translate the pages
7. optional: *modify Login.jsp* for the new locale *xx_YY*
8. compress/zip the working directory and create a new file **opencrx-core-CRX.war** (on the Windows platform you can use Winzip, *7zip (http://sourceforge.net/projects/sevenzip/)*, etc.)
9. create a new file **opencrx-core-CRX-web.ear** which can be deployed to the application server (on the Windows platform you can use Winzip, *7zip (http://sourceforge.net/projects/sevenzip/)*, etc.)

Subsequently, we will look at each of the steps required to create language-specific files in more detail.

Tip

We do not recommend to edit the language-specific files of your openCRX installation as you risk messing up your installation if anything goes wrong. We rather recommend to copy the file *opencrx-core-CRX.war* to a working/temporary directory and then explode/unzip it to get easy access to all the language-specific files.

For this guide it is assumed that

- an *openCRX* core distribution was set up for Ant in the directory **C:\opencrx** (as described in the README of *openCRX*, including the installation of openMDX) and that
- the file *opencrx-core-CRX.war* was exploded/unzipped to the directory **C:\temp\opencrx-core-CRX.war** yielding the following directory structure:

Example 4-1. directory structure of the exploded/unzipped file opencrx-core-CRX.war

```
C:\temp\opencrx-core-CRX.war
|  ...
\--WEB-INF
   +--config
   |  +--code
   |  |  +--de_CH
   |  |  \--en_US
   |  |  \--...
   |  +--texts
   |  |  +--de_CH
   |  |  \--en_US
   |  |  \--...
   |  \--ui
   |  |  +--de_CH
   |  |  \--en_US
   |  |  \--...
   |  ...
```

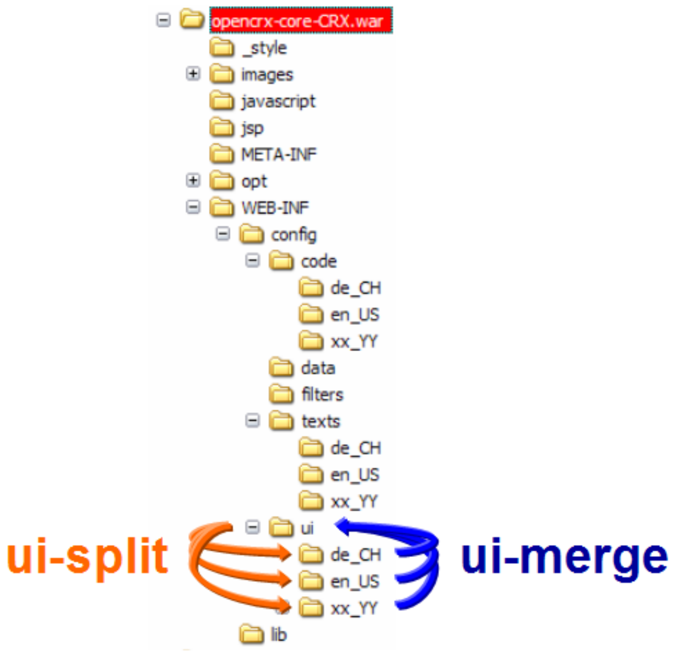
user interface configuration files

For advanced *openCRX* administrators there is a fast procedure to create *user interface configuration files* for a new locale *xx_YY*:

- copy an existing locale directory (e.g. *de_CH*) with all its files and rename the copied directory to *xx_YY*
- translate all the label and tooltip strings in the user interface configuration files

This "manual" management of localized files, however, makes it rather difficult to maintain consistency across multiple locales if the user interface customization changes. The tools **ui-merge** and **ui-split** exist to help you manage and maintain a large number of locales. With *ui-merge* you can pull together information from several locales and create XML files that are easy to edit, with *ui-split* you can push the relevant information back into the respective locale files.

Figure 4-1. ui-merge and ui-split



The automated approach with *ui-merge* and *ui-split* offers the following advantages:

- locales can be managed individually in their respective subdirectories (e.g. the files for locale *de_CH* are located in a directory named **de_CH**)
- it is easy to add/remove locales and you can manage a very large number of locales in a convenient way
- configuring locales for *openCRX* is a matter of manually editing 1 file only: *web.xml*
- migrating to new versions of *openCRX* does not destroy/damage existing locales
- the use of *ui-merge* and *ui-split* ensures that all locale files conform to a standard structure (i.e. are schema-validated)

Furthermore, *ui-merge* and *ui-split* support **locale-migration** as follows:

- if a new version of *openCRX* contains new elements it is easy to spot the "gaps" in a merged user interface configuration file
- removed elements (which may still be present in some of your personalized locale files) are protocolled by *ui-merge*
- all operations by *ui-merge* and *ui-split* create schema-validated files and ensure consistency

Hence, we strongly encourage you to use the provided tools *ui-merge* and *ui-split* and do not recommend to edit individual files manually unless you know exactly what you are doing.

So, let's get started. In a first step, you run **ui-merge** to create *merged user interface configuration files* containing place holders for your new locale *xx_YY* (and optionally other locales in addition to the default locale *en_US* to have more "examples" available for the translation process). Then you edit the merged user interface configuration files to add the translated strings for the new locale *xx_YY*. Once you are done with the translations you run **ui-split** to extract the relevant files for each locale from the *merged user interface configuration files*.

The following steps will guide you through the process of creating the user interface configuration files for the new locale *xx_YY*:

- Open a new command shell and set the current directory to **C:\temp\opencrx-core-CRX.war\config\ui**
- Enter the command **ant -f C:\opencrx\core\build.xml ui-merge -Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY** and execute it to create *merged user interface configuration files* containing place holders for your new locale *xx_YY* (don't forget to adapt the path to the file *utilities.xml* in the command line above to reflect your local installation directory! In a Linux/Unix environment you have to replace **%CD%** with **$PWD** to indicate the current directory)

ui-merge does not overwrite existing files in the target directory. In case of a conflict the existing file is rename by prepending the string **.#** to the name of the file. To start with a clean slate it is a good idea to delete all the files in the directory **C:\temp\opencrx-core-CRX.war\config\ui** before executing ui-merge (but do not delete the subdirectories or the files in the subdirectories!).

Example 4-2. example output of ui-merge

```
C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui>ant -f C:\opencrx\core\build.xml ui-merge -
Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY
Buildfile: C:\opencrx\core\build.xml

ui-merge:
     [java] sourceDir= C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\en_US\abstractcontract.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\abstractcontract.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\en_US\account.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\account.xml
     ...
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\en_US\wf.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\wf.xml
     [java] shutdown

BUILD SUCCESSFUL
Total time: 6 seconds
```

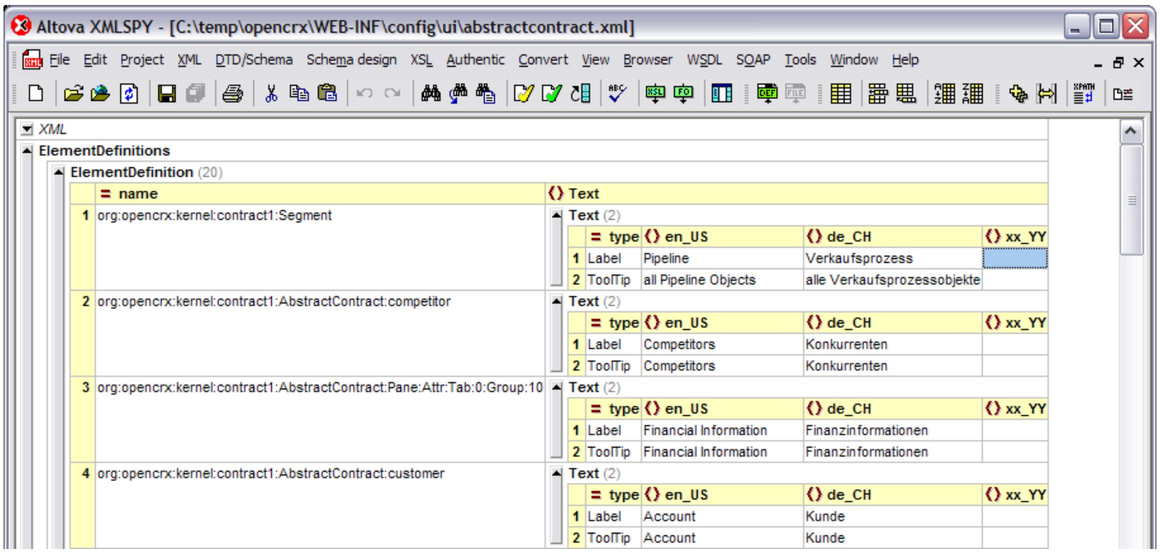
ui-merge creates *merged user interface configuration files* in the directory **C:\temp\opencrx-core-CRX.war\config\ui**. These merged files contain all the ElementDefinitions with labels and/or toolTips:

Example 4-3. example ElementDefinition containing information for the locale en_US (default, always included) and gaps for the newly created locale xx_YY

```
...
<ElementDefinition name="org:opencrx:kernel:contract1:Segment">
    <Text type="Label">
        <en_US>Pipeline</en_US>
        <xx_YY></xx_YY>
    </Text>
    <Text type="ToolTip">
        <en_US>all Pipeline Objects</en_US>
        <xx_YY></xx_YY>
    </Text>
</ElementDefinition>
...
```

You can now use any editor suitable to edit UTF-8 encoded files to add the translations for the locale *xx_YY*. You simply insert the translated string between the opening tag **<xx_YY>** and the closing tag **</xx_YY>**. If you have access to an xml editor that features a grid view (e.g. *XMLfox (http://www.xmlfox.com/)* or xmlspy) it is almost like filling in a spreadsheet.

Figure 4-2. Editing user interface configuration file with xmlspy (locales en_US, de_CH, and xx_YY)


Altova XMLSPY - [C:\temp\opencrx\WEB-INF\config\ui\abstractcontract.xml]
File Edit Project XML DTD/Schema Schema design XSL Authentic Convert View Browser WSDL SOAP Tools Window Help

XML
ElementDefinitions
ElementDefinition (20)

	= name	= type	() en_US	() de_CH	() xx_YY
1	org:opencrx:kernel:contract1:Segment	Label	Pipeline	Verkaufsprozess	
		ToolTip	all Pipeline Objects	alle Verkaufsprozessobjekte	
2	org:opencrx:kernel:contract1:AbstractContract:competitor	Label	Competitors	Konkurrenten	
		ToolTip	Competitors	Konkurrenten	
3	org:opencrx:kernel:contract1:AbstractContract:Pane:Attr:Tab:0:Group:10	Label	Financial Information	Finanzinformationen	
		ToolTip	Financial Information	Finanzinformationen	
4	org:opencrx:kernel:contract1:AbstractContract:customer	Label	Account	Kunde	
		ToolTip	Account	Kunde	

Even though xmlspy is also available in a free edition (*xmlspy Home Edition (http://www.altova.com/download_spy_home.html)*) which allows you to activate the grid view feature for 1 day, you may want to have a look at some of the following alternatives if you do not want to use a simple text editor supporting UTF-8 encoding:

- *Amaya (http://www.w3.org/Amaya/)* (multi-platform, no grid view feature)
- *XMLmind XML editor (http://www.xmlmind.com/xmleditor/)* (multi-platform, no grid view feature)
- *XMLfox (http://www.xmlfox.com/)* (Windows only, with grid view feature)
- *Cooktop (http://www.xmlcooktop.com/)* (Windows only, no grid view feature)

Important Modifying language-specific files requires an editor capable of handling files in **UTF-8** format. The UTF-8 encoding of an XML file is indicated at the beginning of the file with <?xml version="1.0" encoding="UTF-8"?>. More information about encoding of XML files is available at *http://www.w3schools.com/xml/xml_encoding.asp*. If you modify the user interface configuration files with an editor that cannot handle UTF-8 encoded files properly you risk running into problems when starting *openCRX* with such modified files because the XML importer will not be able to correctly import them.

As you will realize, there are quite a lot of strings to translate. However, you might get away with translating a subset of the existing labels/toolTips by making use of the *Fallback Mechanism* built into *openCRX* in the case where no label/toolTip exists for a particular locale. The fallback mechanism is explained in detail in *Fallback Mechanism*.

Tip It might be helpful to have additional locales available in the *merged user interface configuration files* to make the translation process easier (e.g. if you want to create a French translation it might be helpful to have the English *and* the German versions available in the same file). ui-merge supports merging of multiple locales into a single file. When calling ui-merge you can use the parameter **-Darg.locale** to provide a list of all the locales (put the list in quotes "..." and separate individual locales with slashes, e.g. "aa_BB/cc_DD/ee_FF") that you want to merge (note that there is no need to specify the locale en_US as this default locale is always included). For example, to merge the locales en_US, de_CH, and xx_YY you would use the following command line:

ant -f C:\opencrx\core\build.xml ui-merge -Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale="de_CH/xx_YY"

When calling ui-merge you can use the parameter **-Darg.format=schema** to produce merged files containing all the formatting information as well (there is no need to specify the default **-Darg.format=table**).

Once you are done with the translations you need to extract the relevant files for each locale from the *merged user interface configuration files*. This is done with ui-split.

- Open a new command shell and set the current directory to **C:\temp\opencrx-core-CRX.war\config\ui**
- Enter the command **ant -f C:\opencrx\core\build.xml ui-split -Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY** and execute it to create *individual user interface configuration files for each specified locale* (don't forget to adapt the path to the file *utilities.xml* in the command line above to reflect your local installation directory! In a Linux/Unix environment you have to replace **%CD%** with **$PWD** to indicate the current directory)

Example 4-4. example output of ui-split

```
C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui>ant -f C:\opencrx\core\build.xml ui-split -
Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY
Buildfile: C:\opencrx\core\build.xml

ui-split:
     [java] sourceDir= C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\en_US\abstractcontract.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\abstractcontract.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-
INF\config\ui\xx_YY\abstractcontract.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\en_US\account.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\account.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\xx_YY\account.xml
     ....
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\en_US\wf.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\wf.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\ui\xx_YY\wf.xml
     [java] shutdown

BUILD SUCCESSFUL
Total time: 7 seconds
```

ui-split creates all the required directories and files for the new locale *xx_YY*. In particular, the directory **C:\temp\opencrx-core-CRX.war\config\ui** should now contain a subdirectory **xx_YY** with all the *user interface configuration files* specific to locale *xx_YY*.

ui-split does not overwrite existing files in the target directories. In case of a conflict the existing file is rename by prepending the string **.#** to the name of the file. To start with a clean slate it is a good idea to delete all the files in those subdirectories of **C:\temp\opencrx-core-CRX.war\config\ui** related to the locale(s) you've been modifying before executing ui-merge (but do not delete the files in the directory **C:\temp\opencrx-core-CRX.war\config\ui** and do not delete the files in the base directory **C:\temp\opencrx-core-CRX.war\config\ui\en_US**).

code table files

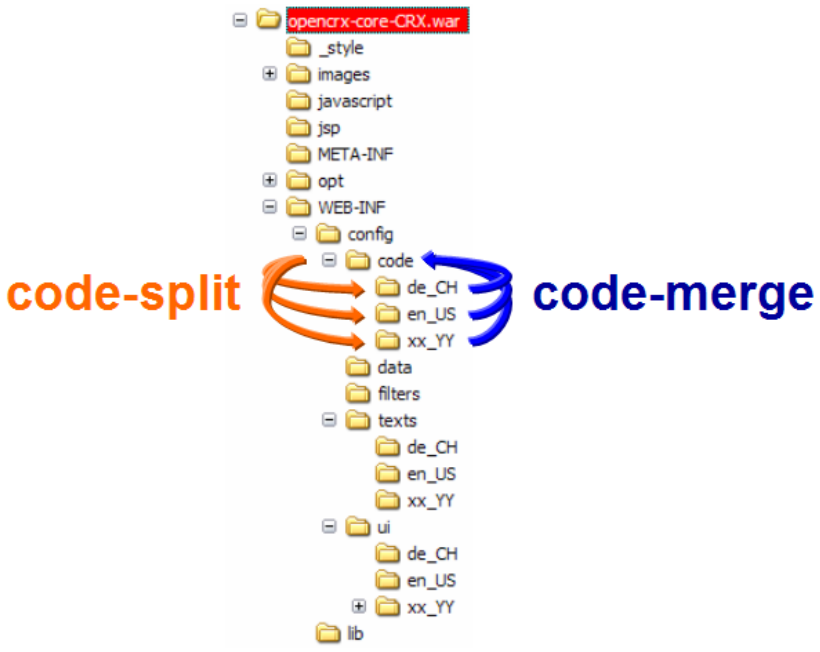
For advanced *openCRX* administrators there is a fast procedure to create *code table files* for a new locale *xx_YY*:

- add a new entry to the file **C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\en_US\locale.xml** reflecting the new locale *xx_YY*
- copy an existing locale directory (e.g. *de_CH*) with all its files and rename the copied directory to *xx_YY*
- translate all texts in the code table files

Important You must ensure that the **index number in locale.xml matches the index number in web.xml for this newly added locale** (e.g. if *xx_YY* is assinged to index number *m* in locale.xml then *xx_YY* must be assigned to the index number *m* in web.xml). See also *Configuring locales for openCRX with web.xml*

This "manual" management of localized files, however, makes it rather difficult to maintain consistency across multiple locales if code tables are modified/extended. The tools **code-merge** and **code-split** exist to help you manage and maintain a large number of locales. With *code-merge* you can pull together information from several locales and create XML files that are easy to edit, with *code-split* you can push the relevant information back into the respective locale files.

Figure 4-3. code-merge and code-split



The automated approach with *code-merge* and *code-split* offers the following advantages:

- locales can be managed individually in their respective subdirectories (e.g. the files for locale *de_CH* are located in a directory named **de_CH**)
- it is easy to add/remove locales and you can manage a very large number of locales in a convenient way
- configuring locales for *openCRX* is a matter of manually editing 1 file only: *web.xml*
- migrating to new versions of *openCRX* does not destroy/damage existing locales
- the use of *code-merge* and *code-split* ensures that all locale files conform to a standard structure (i.e. are schema-validated)

Furthermore, *code-merge* and *code-split* support **locale-migration** as follows:

- if a new version of *openCRX* contains new codes it is easy to spot the "gaps" in a merged code table file
- removed codes (which may still be present in some of your personalized locale files) are protocolled by *code-merge*
- all operations by *code-merge* and *code-split* create schema-validated files and ensure consistency

Hence, we strongly encourage you to use the provided tools *code-merge* and *code-split* and do not recommend to edit individual files manually unless you know exactly what you are doing.

In a first step, you run **code-merge** to create *merged code table files* containing place holders for your new locale *xx_YY* (and optionally other locales in addition to the default locale *en_US* to have more "examples" available for the translation process). Then you edit the merged code table files to add the translated strings for the new locale *xx_YY*. Once you are done with the translations you run **code-split** to extract the relevant files for each locale from the *merged code table files*.

The following steps will guide you through the process of creating the code table files for the new locale *xx_YY*:

- add a new entry to the file **C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\en_US\locale.xml** reflecting the new locale *xx_YY*
- Open a new command shell and set the current directory to **C:\temp\opencrx-core-CRX.war\config\code**
- Enter the command **ant -f C:\opencrx\core\build.xml code-merge -Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY** and execute it to create *merged code table files* containing place holders for your new locale *xx_YY* (don't forget to adapt the path to the file *utilities.xml* in the command line above to reflect your local installation directory! In a Linux/Unix environment you have to replace **%CD%** with **$PWD** to indicate the current directory)

Important code-merge does not overwrite existing files in the target directory. In case of a conflict the existing file is rename by prepending the string **.#** to the name of the file. To start with a clean slate it is a good idea to delete all the files in the directory **C:\temp\opencrx-core-CRX.war\config\code** before executing code-merge (but do not delete the subdirectories or the files in the subdirectories!).

Example 4-5. example output of code-merge

```
C:\temp\opencrx-core-CRX.war\WEB-INF\config\code>ant -f C:\opencrx\core\build.xml code-merge -
Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY
Buildfile: C:\opencrx\core\build.xml

code-merge:
     [java] sourceDir=C:\temp\opencrx-core-CRX.war\WEB-INF\config\code
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\en_US\accountcategory.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\accountcategory.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\en_US\accountstate.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\accountstate.xml
     ...
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\en_US\utcoffset.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\utcoffset.xml
     [java] shutdown

BUILD SUCCESSFUL
Total time: 7 seconds
```

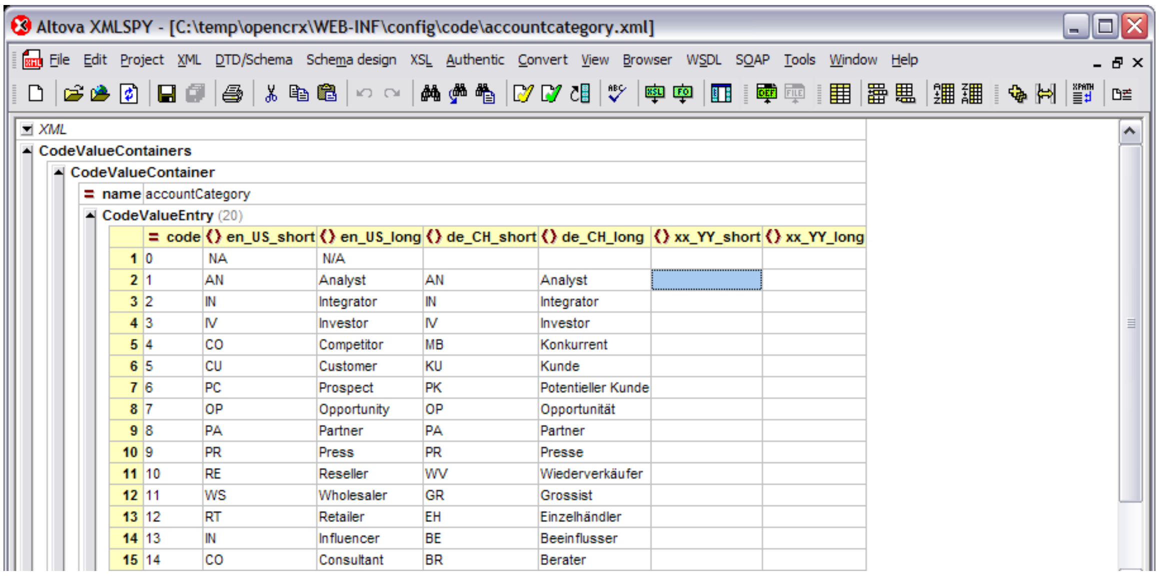
code-merge creates *merged code table files* in the directory **C:\temp\opencrx-core-CRX.war\config\code**. These merged files contain all the CodeValueContainers with CodeValueEntries:

Example 4-6. example CodeValueEntry containing information for the locale en_US (default, always included) and gaps for the newly created locale xx_YY

```
...
<CodeValueContainer name="accountCategory">
    <CodeValueEntry code="0">
        <en_US_short> NA</en_US_short>
        <en_US_long> N/A</en_US_long>
        <xx_YY_short></xx_YY_short>
        <xx_YY_long></xx_YY_long>
    </CodeValueEntry>
    …
...
```

You can now use any editor suitable to edit UTF-8 encoded files to add the translations for the locale *xx_YY*. You simply insert the translated string between the opening tag **<xx_YY>** and the closing tag **</xx_YY>**. If you have access to an xml editor that features a grid view (e.g. *XMLfox (http://www.xmlfox.com/)* or xmlspy) it is almost like filling in a spreadsheet.

Figure 4-4. Editing code table file with xmlspy (locales en_US, de_CH, and xx_YY)


Altova XMLSPY - [C:\temp\opencrx\WEB-INF\config\code\accountcategory.xml]

	code	en_US_short	en_US_long	de_CH_short	de_CH_long	xx_YY_short	xx_YY_long
1	0	NA	N/A				
2	1	AN	Analyst	AN	Analyst		
3	2	IN	Integrator	IN	Integrator		
4	3	IV	Investor	IV	Investor		
5	4	CO	Competitor	MB	Konkurrent		
6	5	CU	Customer	KU	Kunde		
7	6	PC	Prospect	PK	Potentieller Kunde		
8	7	OP	Opportunity	OP	Opportunität		
9	8	PA	Partner	PA	Partner		
10	9	PR	Press	PR	Presse		
11	10	RE	Reseller	WV	Wiederverkäufer		
12	11	WS	Wholesaler	GR	Grossist		
13	12	RT	Retailer	EH	Einzelhändler		
14	13	IN	Influencer	BE	Beeinflusser		
15	14	CO	Consultant	BR	Berater		

Important Modifying language-specific files requires an editor capable of handling files in **UTF-8** format. The UTF-8 encoding of an XML file is indicated at the beginning of the file with <?xml version="1.0" encoding="UTF-8"?>. More information about encoding of XML files is available at *http://www.w3schools.com/xml/xml_encoding.asp*. If you modify the code table files with an editor that cannot handle UTF-8 encoded files properly you risk running into problems when starting *openCRX* with such modified files because the XML importer will not be able to correctly import them.

As you will realize, there are quite a lot of strings to translate. However, you might get away with translating a subset of the existing code values by making use of the *Fallback Mechanism* built into *openCRX* in the case where no code value string exists for a particular locale. The fallback mechanism is explained in detail in *Fallback Mechanism*.

It might be helpful to have additional locales available in the *merged code table files* to make the translation process easier (e.g. if you want to create a French translation it might be helpful to have the English *and* the German versions available in the same file). code-merge supports merging of multiple locales into a single file. When calling code-merge you can use the parameter **-Darg.locale** to provide a list of all the locales (put the list in quotes "..." and separate individual locales with slashes, e.g. "aa_BB/cc_DD/ee_FF") that you want to merge (note that there is no need to specify the locale en_US as this default locale is always included). For example, to merge the locales en_US, de_CH, and xx_YY you would use the following command line:

ant -f C:\opencrx\core\build.xml code-merge -Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale="de_CH/xx_YY"

When calling code-merge you can use the parameter **-Darg.format=schema** to produce merged files containing all the formatting information as well (there is no need to specify the default **-Darg.format=table**).

Once you are done with the translations you need to extract the relevant files for each locale from the *merged code table files*. This is done with code-split.

- Open a new command shell and set the current directory to **C:\temp\opencrx-core-CRX.war\config\code**
- Enter the command **ant -f C:\opencrx\core\build.xml code-split -Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY** and execute it to create *individual code table files for each specified locale* (don't forget to adapt the path to the file *utilities.xml* in the command line above to reflect your local installation directory! In a Linux/Unix environment you have to replace **%CD%** with **$PWD** to indicate the current directory)

Example 4-7. example output of code-split

```
C:\temp\opencrx-core-CRX.war\WEB-INF\config\code>ant -f C:\opencrx\core\build.xml code-split -
Darg.sourceDir=%CD% -Darg.targetDir=%CD% -Darg.locale=xx_YY
Buildfile: C:\opencrx\core\build.xml

code-split:
     [java] sourceDir=C:\temp\opencrx-core-CRX.war\WEB-INF\config\code
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\en_US\accountcategory.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\accountcategory.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\xx_YY\accountcategory.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\en_US\accountstate.xml
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\accountstate.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\xx_YY\accountstate.xml
     ...
     [java] loading C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\utcoffset.xml
     [java] writing file C:\temp\opencrx-core-CRX.war\WEB-INF\config\code\xx_YY\utcoffset.xml
     [java] shutdown

BUILD SUCCESSFUL
Total time: 7 seconds
```

code-split creates all the required directories and files for the new locale *xx_YY*. In particular, the directory **C:\temp\opencrx-core-CRX.war\config\code** should now contain a subdirectory **xx_YY** with all the *code table files* specific to locale *xx_YY*.

code-split does not overwrite existing files in the target directories. In case of a conflict the existing file is rename by prepending the string **.#** to the name of the file. To start with a clean slate it is a good idea to delete all the files in those subdirectories of **C:\temp\opencrx-core-CRX.war\config\code** related to the locale(s) you've been modifying before executing code-merge (but do not delete the files in the directory **C:\temp\opencrx-core-CRX.war\config\code** and do not delete the files in the base directory **C:\temp\opencrx-core-CRX.war\config\code\en_US**).

texts.properties files

The files *opencrx.text.properties* and *text.properties* contain several strings used by JSP. These files must exist for each locale configured in *openCRX*. The following steps will guide you through the process of creating the files *opencrx.text.properties* and *text.properties* for the new locale *xx_YY*:

- Create the new directory **C:\temp\opencrx-core-CRX.war\config\texts\xx_YY**
- Decide which locale you want to use as a template (starting point) for your new locale *xx_YY*. In this example we will use *en_US* as a template. Hence, copy the files *opencrx.text.properties* and *text.properties* from the directory **C:\temp\opencrx-core-CRX.war\config\texts\en_US** to the newly created directory **C:\temp\opencrx-core-CRX.war\config\texts\xx_yy**
- Open the file *text.properties* in the directory **C:\temp\opencrx-core-CRX.war\config\texts\xx_YY** with a text editor.
- Each line of the file contains a name-value-pair of the form **name**=**value**. Replace the existing values with your translations and then save the file *text.properties* with all the changes.
- Open the file *opencrx.text.properties* in the directory **C:\temp\opencrx-core-CRX.war\config\texts\xx_YY** with a text editor.
- Each line of the file contains a name-value-pair of the form **name**=**value**. Replace the existing values with your translations and then save the file *opencrx.text.properties* with all the changes.

Important Use ISO 8859-1 character encoding for the files *opencrx.text.properties* and *text.properties* (the reason being that *openCRX* relies on the class java.util.Properties of the Java™ 2 platform). For characters that cannot be directly represented in this encoding, Unicode escapes are used; however, only a single 'u' character is allowed in an escape sequence. The JDK native2ascii tool can be used to convert property files to and from other character encodings (e.g. to convert a file *texts.properties.uni* to *text.properties* (ISO 8859-1 encoded) you could run *native2ascii -encoding utf-8 texts.properties.uni texts.properties*)

Example 4-8. extract from the file ...\en_US\text.properties – each line containing a name-value-pair

```
Locale=en_US
dir=ltr
LocaleTitle=English (United States)
CancelTitle=Cancel
OkTitle=OK
SaveTitle=Save
SortAscendingText=Click to sort ascending
SortDescendingText=Click to sort descending
DisableSortText=Click to disable sorting
DeleteTitle=Delete
EditTitle=Edit
...
```

If you don't feel like translating all the values, you can just leave the template values unchanged. However, it is important that the files *opencrx.text.properties* and *text.properties* exist for all configured locales and the files must contain all name-value-pairs.

html help pages

openCRX html help pages are located in the directory **C:\temp\opencrx-core-CRX.war**. The following steps will guide you through the process of creating the html pages for the new locale *xx_YY*:

- Decide which locale you want to use as a template (starting point) for your new locale *xx_YY*. In this example we will use *en_US* as a template. Hence, make a copy the file *helpSearch_en_US.html* and name it *helpSearch_xx_YY.html*
- Open the file *helpSearch_xx_YY.html* with an html editor (a simple text editor will also work if you are familiar with html).
- Once you are done with translating the whole page you save the file *helpSearch_xx_YY.html* with all the changes.

Repeat the above steps for the rest of the *openCRX* html help pages located in the directory **C:\temp\opencrx-core-CRX.war.**

Important If you don't feel like translating the help pages (or not all of them) you must still create the html pages for the new locale *xx_YY* as described above because there is no fallback mechanism for html pages – missing pages will cause an error 404 (page not found) if a user requests such a page.

modify Login.jsp

The *openCRX* login page *Login.jsp* is located in the directory **C:\temp\opencrx-core-CRX.war**. The login page is a special case because prior to authentication the user does not have access to the *openCRX* application (and hence the *openCRX* localization features are not available). However, you only need to modify the login page if you want to offer your users a login page in a locale not already configured. You can add a new locale *xx_YY* by extending the code at the appropriate positions (e.g. by searching for "en_US" or "de_CH" which are both locales built into Login.jsp).

The language specific part of *Login.jsp* starts with the following lines:

Example 4-9. extract from the file ...\opencrx-core-CRX.war\Login.jsp

```
// test whether requested locale is supported
if((locale == null) ||
  (!locale.equals("en_US") &&
   !locale.equals("de_CH") &&
   !locale.equals("es_MX") &&
   …
```

Extend all the hash maps following the above code sequence so that the updated login page can fully support the new locale *xx_YY*.

Tip

If a login page supports locale *xx_YY* you can request the login page in that locale xx_YY by appending the string "**?locale=xx_YY**" to the default login URL._Example: the URL **http://demo.opencrx.org/opencrx-core-CRX/Login?locale=de_CH** directly loads the German login page.

Chapter 5. Configuring locales for openCRX with web.xml

Configuring locales for *openCRX* is done by modifying the file *web.xml* in the directory **C:\temp\opencrx-core-CRX.war\WEB-INF**. If you open the file web.xml you will find the following section:

Example 5-1. web.xml containing locale configuration information

```
…
<!-- locales -->
    <init-param>
        <param-name>locale[0]</param-name>
        <param-value>en_US</param-value>
    </init-param>
    <init-param>
        <param-name>locale[1]</param-name>
        <param-value>de_CH</param-value>
    </init-param>
…
```

To configure an additional locale for *openCRX* you need to add a new locale-block. For example, to add the locale *xx_YY* you modify *web.xml* as follows:

Example 5-2. web.xml containing locale configuration information for en_US, de_CH, and the new locale xx_YY

```
…
<!-- locales -->
    <init-param>
        <param-name>locale[0]</param-name>
        <param-value>en_US</param-value>
    </init-param>
    <init-param>
        <param-name>locale[1]</param-name>
        <param-value>de_CH</param-value>
    </init-param>
   <init-param>
        <param-name>locale[2]</param-name>
        <param-value>xx_YY</param-value>
    </init-param>
…
```

Important Please note that the **index number must be unique** (i.e. you can only assign locale *xx_YY* to index number "2" with locale[2] if no other locale is assigned to index number "2"). Furthermore, **index number "0" is reserved for the default locale** (by default, locale *en_US* is assigned to index number "0").

Important You must ensure that the **index number in web.xml matches the index number in locale.xml for this newly added locale** (e.g. if *xx_YY* is assinged to index number *m* in locale.xml then *xx_YY* must be assigned the index number *m* in web.xml). See also *code table files*.

Important When configuring locales for *openCRX* it is important that you are aware of the *Fallback Mechanism*. Taking into account what happens when there is no localized directory/file/string/entry/etc. available for a configured locale helps you to control what the user of *openCRX* is going to see in place of a nicely translated text.

Fallback Mechanism

The following rules apply for missing localization information (it is assumed that the locale is *xx_YY*):

user interface configuration

if no entry is found for the respective label/toolTip, then the existing entry of the locale xx_ZZ (i.e. same language as xx_YY) with the highest index number (see Configuring locales for openCRX with web.xml) is taken. if no entry is found with the same language xx, then the entry of the default locale (i.e. the locale with index number "0", usually en_US) is taken.

code tables

if no entry is found for the respective code, then the existing entry of the locale xx_ZZ (i.e. same language as xx_YY) with the highest index number (see Configuring locales for openCRX with web.xml) is taken. if no entry is found with the same language xx, then the entry of the default locale (i.e. the locale with index number "0", usually en_US) is taken.

opencrx.texts.properties and **texts.properties**

if no entry is found for the respective name, then the existing entry of the locale xx_ZZ (i.e. same language as xx_YY) with the highest index number (see Configuring locales for openCRX with web.xml) is taken. if no entry is found with the same language xx, then the entry of the default locale (i.e. the locale with index number "0", usually en_US) is taken.

html help pages

there is no fallback mechanism – if a page is missing for a requested locale the user will get an error 404 (page not found)

login.jsp

there is no fallback mechanism – login.jsp is a special case anyway (see *modify Login.jsp*)

The following description explains the implemented locale fallback mechanism of *openCRX* in a somewhat more formal way (only applies to user interface configuration, code tables, and text.properties). Please note that the locale fallback is based on language *xx* and not on a fully qualified locale string *xx_YY*:

- Iterate all locales configured in web.xml starting with index number 0, i.e. locale[0], locale[1], …
- Try to load resources from locale *i*, i.e. config/code/<locale *i*>, config/ui/<locale *i*>, config/texts/<locale *i*>
- If no resources are found at locale *i* fallback to resources of locale *j*, where *j* = max(*J*), *J* = {0} + {*j*, locale[*i*].lang == locale[*j*].lang}

Example: If e.g. the locales **en_US** (locale[0]), **de_CH** (locale[1]) and **de_DE** (locale[2]) are configured and no resources are available for **de_DE** then **de_DE** falls back to **de_CH**.

Chapter 6. Next Steps

Appendix A. Appendix

Bibliography

[01] *openCRX - the leading open source CRM solution*, opencrx.org.

@ *http://www.opencrx.org* (http://www.opencrx.org)

[02] *openMDX - The leading open source MDA platform*, openmdx.org.

@ *http://www.openmdx.org* (http://www.openmdx.org)