

Internationalizing and Localizing Rhythmyx

5.7

Copyright and Licensing Statement

All intellectual property rights in the SOFTWARE and associated user documentation, implementation documentation, and reference documentation are owned by Percussion Software or its suppliers and are protected by United States and Canadian copyright laws, other applicable copyright laws, and international treaty provisions. Percussion Software retains all rights, title, and interest not expressly granted. You may either (a) make one (1) copy of the SOFTWARE solely for backup or archival purposes or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You must reproduce and include the copyright notice on any copy made. You may not copy the user documentation accompanying the SOFTWARE.

The information in Rhythmyx documentation is subject to change without notice and does not represent a commitment on the part of Percussion Software, Inc. This document describes proprietary trade secrets of Percussion Software, Inc. Licensees of this document must acknowledge the proprietary claims of Percussion Software, Inc., in advance of receiving this document or any software to which it refers, and must agree to hold the trade secrets in confidence for the sole use of Percussion Software, Inc.

Copyright © 1999-2005 Percussion Software.
All rights reserved

Licenses and Source Code

Rhythmyx uses Mozilla's JavaScript C API. See <http://www.mozilla.org/source.html> (<http://www.mozilla.org/source.html>) for the source code. In addition, see the *Mozilla Public License* (<http://www.mozilla.org/source.html>).

Netscape Public License

Apache Software License

IBM Public License

Lesser GNU Public License

Other Copyrights

The Rhythmyx installation application was developed using InstallShield, which is a licensed and copyrighted by InstallShield Software Corporation.

The Sprinta JDBC driver is licensed and copyrighted by I-NET Software Corporation.

The Sentry Spellingchecker Engine Software Development Kit is licensed and copyrighted by Wintertree Software.

The Java™ 2 Runtime Environment is licensed and copyrighted by Sun Microsystems, Inc.

The Oracle JDBC driver is licensed and copyrighted by Oracle Corporation.

The Sybase JDBC driver is licensed and copyrighted by Sybase, Inc.

The AS/400 driver is licensed and copyrighted by International Business Machines Corporation.

The Ephox EditLive! for Java DHTML editor is licensed and copyrighted by Ephox, Inc.

This product includes software developed by CDS Networks, Inc.

The software contains proprietary information of Percussion Software; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited.

Due to continued product development this information may change without notice. The information and intellectual property contained herein is confidential between Percussion Software and the client and remains the exclusive property of Percussion Software. If you find any problems in the documentation, please report them to us in writing. Percussion Software does not warrant that this document is error-free.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of Percussion Software.

AuthorIT™ is a trademark of Optical Systems Corporation Ltd.

Microsoft Word, Microsoft Office, Windows®, Window 95™, Window 98™, Windows NT® and MS-DOS™ are trademarks of the Microsoft Corporation.

This document was created using *AuthorIT™, Total Document Creation* (see AuthorIT Home - <http://www.author-it.com>).

Percussion Software

600 Unicorn Park Drive

Woburn, MA 01801 U.S.A

781.438.9900

Internet E-Mail: technical_support@percussion.com

Website: <http://www.percussion.com>

Contents

| | |
|--|----------|
| Introduction to Internationalization and Localization in Rhythmyx | 3 |
|--|----------|

| | |
|--|----------|
| Determining System Requirements | 5 |
|--|----------|

| | |
|--|----------|
| Modeling Internationalization of a Rhythmyx Content Management System | 7 |
|--|----------|

| | |
|---|----|
| Modeling Assembly for Internationalization | 8 |
| Modeling Internationalized Workflows | 9 |
| Defining the Content Relationship Model for Translations..... | 9 |
| Modeling Internationalized Workflow Processes | 9 |
| Spinning Off Translation Versions of Content Items | 10 |
| Modeling Publishing..... | 11 |

| | |
|--|-----------|
| Implementing Internationalization | 13 |
|--|-----------|

| | |
|--|----|
| Internationalization Support on the Rhythmyx Server..... | 14 |
| Localizing the EditLive Text Editor | 17 |
| Implementing Internationalized Assembly | 18 |
| Implementing Internationalization of Variants..... | 18 |
| Implementing Internationalized Numbers and Date Formats | 19 |
| Changing Output Character Sets | 21 |
| Creating Automatic Links to Other Localized Versions of Content..... | 21 |
| Localizing the Business User Interface..... | 22 |
| Localizing the Interface | 22 |
| Preparing XSL Files for Localization..... | 27 |
| Localizing JavaScript Files..... | 29 |
| Using Locale-specific Images | 31 |
| Defining Translation Properties | 32 |
| Auto Translation Edit Configuration Page | 32 |
| Adding a Translation Configuration | 33 |
| Editing a Translation Configuration | 33 |
| Deleting a Translation Configuration | 33 |
| Implementing Automated Translation | 34 |
| Example Auto Translation Implementation..... | 34 |
| Configuring the sys_createTranslations Workflow Action | 38 |
| Implementing Internationalized Publishing | 39 |
| Implementing Language-centric Publishing Models | 39 |
| Implementing Content-centric Publishing Models | 39 |
| Full-text Search in Globalized Environments..... | 40 |

| | |
|--------------|-----------|
| Index | 41 |
|--------------|-----------|

CHAPTER 1

Introduction to Internationalization and Localization in Rhythmyx

Internationalizing and Localizing Rhythmyx explains how to model a Rhythmyx Content Management System (CMS) for internationalization and then how to implement your model. The document assumes that you are developing a new system for internationalization. It also assumes that you are generally familiar with the process of modeling and implementing a non-internationalized CMS, either through training on Rhythmyx or through reading the following Rhythmyx documentation:

- *Modeling a Content Management System*
- *Implementing Content Editors and Assemblers*
- *Implementing Publishing*
- *Implementing the Content Contributor Interface*
- *Maintaining Workflows*

If you are not familiar with these documents, you should read them before proceeding with this document.

Internationalization is the process of implementing a Rhythmyx CMS to support content in multiple languages. The Rhythmyx Content Model provides a firm foundation for internationalization because it provides you with control over each item. Rhythmyx has functionality that allows you to determine which items to translate, and the languages into which to translate them. You can implement your system to create translation Versions of items automatically or allow your users to determine whether an item should be translated.

Rhythmyx uses the Java definition of Locale, which identifies a combination of language and geographical, political, and cultural region. For more details about internationalization in Java, see <http://java.sun.com/products/jdk/1.1/docs/guide/intl/>.

CHAPTER 2

Determining System Requirements

When internationalizing Rhythmyx, the only special requirement you need to consider is the character set of your database. The default character set for Rhythmyx is UTF-8, which is generally adequate for most European languages. If you will store Asian languages that require multiple-bit characters, such as Korean or Japanese, you should install your database with support for UTF-16. It is best to install the database with support for the correct character set from the beginning. Converting a database to a different character set is possible, but very complicated.

CHAPTER 3

Modeling Internationalization of a Rhythmyx Content Management System

Internationalization of a Rhythmyx content management system involves three of the four engines in Rhythmyx, Assembly, Workflow, and Publishing. You do not need to modify your content model to accommodate internationalization.

Implementing internationalization is much easier if you model your internationalized CMS. You will then save time implementing the model.

Modeling Assembly for Internationalization

When modeling internationalization of Content Assembly, Variants fall into three groups:

- Variants that need no modification;
- Variants you can internationalize by customizing the XSL to add localization processing;
- Variants that require a unique XSL template for each Locale.

A number of assemblers will not need modification. Generally, assemblers that produce small, simple snippets, such as title links used in navigation or abstracts used on index pages, do not require localization. They consist of a few lines of text from managed content, with no static text or images, and thus require no special modification for localization.

Variants require internationalization if they include static text or if the template differs significantly for one or more Locales. A Variant that includes images may also require internationalization, depending on the content of the images. If an image includes text or culture-specific content, you probably need to internationalize the Variant that uses that image.

You can offload internationalization of static text in Variant XSL files to the server. You must format the internationalized XSL to make it available for server localization. For details, see “*Preparing XSL Files for Localization* (on page 33)” below.

Another approach for static text is to use conditional processing. Conditional processing is necessary for internationalized images. The conditional processing evaluates the Locale of the XML content and processes the template accordingly, displaying the correct text or retrieving the correct image.

If the Variants differ significantly for different Locales, such as using different table layouts, you probably need to develop Locale-specific Variants. If you have a significant amount of conditional processing, it may also be easier to manage unique Variants for each Locale rather than an overly complicated XSL template.

You can combine both approaches to internationalizing Variants. You may have sets of Locales that can use common images, for example, but that also use unique static text for specific Locales within the Group. For example, you may have several Spanish Locales that use the same set of images, but that use different static text. In that case, you might develop a unique XSL for your Spanish Locale that also includes internal conditional processing to display the appropriate text.

Modeling Internationalized Workflows

Internationalizing Workflows involves three steps:

- Defining the Relationship dependency model between original and translation content items;
- Defining the model for the Workflow processes
- Deciding when to spin off translation Versions of content items.

Defining the Content Relationship Model for Translations

When a user creates a translation Version of a Content Item, the translation Version has a Relationship association with the original item. In many cases, the two items proceed through their respective Workflow processes, become Public, and are published independent of one another. This association is referred to as a *weak dependency*.

In other cases, items from multiple Locales must become Public together. For example, in some jurisdictions, content must be available in multiple languages; it would be illegal to publish a content item in one language if the associated translation was not complete. Both items must be ready to enter a Public State before either can make the Transition. This association is referred to as a *strong dependency*.

Your circumstances determine the model you choose. As indicated above, in some cases, regulations dictate a strong dependency model. Business requirements may also dictate a strong dependency model if your priority is to keep the content on all sites synchronized. A strong dependency model may entail some delay in publishing the content, however, while translation is completed.

Much of the time you have more flexibility about keeping content synchronized. If it is not important that content for all Locales stay synchronized, a weak dependency model may be more appropriate. A weak dependency model may be more appropriate if timely publication of the original content is more important than synchronization of content for all Locales, such as delivering breaking news.

Relationship dependencies can be defined by content type, so you do not have to use a pure model. For example, a system using strong dependencies may include “quick edit” States so you can fix minor mistakes, such as misspellings. A misspelling in an original item is unlikely to affect its translation, so there’s no reason that the translation item should change States while the original is edited.

Modeling Internationalized Workflow Processes

To internationalize Workflow processes, decide whether you need any Locale-specific Workflows, or whether a single translation Workflow will suit your needs. In most cases, a single translation Workflow should be sufficient. You may need Locale-specific Workflows if a Locale has specific translation requirements that differ from other Locales.

Workflow Processes in Strong Dependency Models

In systems that use a strong dependency model, you must pay special attention to your Workflow Processes. A strong dependency model may require that all content enter a Public State together. If you have not designed your Workflow process and your content Relationships correctly, content can become trapped in the Workflow, unable to progress to a Public State.

Best Practice when designing Workflow processes for a strong dependency model is to create a “pending” State prior to the Public State in each Workflow. In this State, all work is effectively done on the Content Item and it is ready to Transition to Public, but must wait for dependent items to reach the same State before they can all make the Transition.

Spinning Off Translation Versions of Content Items

In a strong dependency model, you will need to decide at what phase in the Workflow to spin off the translation Version of the Content Item. Usually, it is best to generate the translation Versions automatically using the `sys_CreateTranslations` Workflow Action. This Action creates a Translation Content Item of the original Content Item in each Locale in which the original Content Item does not already have a corresponding Translation Content Item.

Best Practice is to spin off the translation Versions as late in the Workflow process as possible, ideally when the original item enters the “pending” State. At this point, all revision of the original item is complete, so the translator uses the final version of the original. If you spin off the translation Versions too early, the text of the original may still be changing and the translation may not account for all edits to the original item.

Modeling Publishing

Two delivery models are available for publishing content in Rhythmyx.

- A “language-centric” approach involves creating a file system for each language or modifying your output database with additional columns or tables to store localized content.
- A “content-centric” approach involves creating a single file system or database schema with unique pages or rows for each language.

Both approaches are feasible using Rhythmyx.

If you deliver to remote sites, such as mirrors in the Locale’s country, you should use a language-centric approach rather than a content centric approach. A language-centric approach is also more appropriate if much of the content of your sites is Locale-specific. For example, if you offer different products in different Locales, you probably should use a language-centric approach.

A language-centric approach is also more appropriate if you re-translate content frequently, or if your translators are in-house or otherwise easily accessible. A language-centric approach can be more expensive, however, since it may require multiple servers and databases.

A language-centric delivery model makes it easier to manage all of the content for a single Locale because all of the Locale-specific content resides together. You can also define different Publishing rules for different Locales and administer each Locale separately. Using this model, however, it is difficult to tell whether the content is in sync across Locales (in other words, whether all of the content on the `en-us` site is also on the `fr-ca` site). Sites may be out of sync if you use scheduled Publishing and the schedules for the sites are not synchronized.

If you deliver content from a single server, a content-centric approach may be useful particularly if your content does not differ significantly across Locales. If you translate less frequently (for example, if you outsource translations or if your Localized content is relatively stable), you might prefer a content-centric approach. A content-centric approach to content delivery can also be less expensive because it requires fewer delivery servers and databases.

Using a content-centric approach, it is easier to monitor whether specific content is Published across Locales because content for all Locales is together. However, you cannot define unique publishing rules for different Locales and all Locales are administered together.

If you choose a language-centric approach, you will only need to design the additional file trees or database schemas for your localized sites. You will need to define contexts for each site, as well as global variables and scheme generators to deliver content into each site.

If you choose a content-centric delivery model into a file system, you will need to decide on a naming convention for localized content. A common convention incorporates the Locale identifier into the page name:

```
index_en-us.htm
index_fr-ca.htm
index_de-de.htm
home_en_us.htm
home_fr-ca.htm
home_de-de.htm
```


CHAPTER 4

Implementing Internationalization

This section explains how to implement your internationalization model you developed for Content Assembly, Workflow, and Publishing. Since you do not need to develop a content model for internationalization, no additional implementation is required to internationalize content.

This section also discusses support for internationalization on the Rhythmyx server.

Internationalization Support on the Rhythmyx Server

Several default Rhythmyx CMS tables have been internationalized. You can use these tables in both Content Editors and Content Assemblers. Internationalized tables include:

| Table | Keys |
|-----------------|---|
| COMPONENTS | psx.component.<NAME>@<DISPLAYNAME |
| CONTENTTYPES | psx.contenttype.<CONTENTTYPEID>@<CONTENTTYPENAME> |
| CONTENTVARIANTS | psx.variant.<VARIANTID>@<VARIANTDESCRIPTION> |
| ROLES | psx.role@<ROLENAME> |
| RXCOMMUNITY | psx.community@<NAME> |
| RXLOCALE | psx.locale@<LANGUAGESTRING> |
| RXLOOKUP | psx.keyword.<LOOKUPID>@<LOOKUPDISPLAY> |
| RXMENUACTION | psx.menuitem.<NAME>@<DISPLAYNAME> |
| RXSITES | psx.site.name.<SITEID>@<SITENAME> psx.site.description.<SITEID>@<SITEDESC> |
| RXSLOTTYPE | psx.slot@<SLOTNAME> |
| STATES | psx.workflow.state@<STATENAME> |
| TRANSITIONS | psx.workflow.transition.<WORKFLOWAPPID>.<TRANSITIONID>@<TRANSITION LABEL> |
| WORKFLOWAPPS | psx.workflow.workflow.workflow.<WORKFLOWAPPID>@WORKFLOWAPPNAME |

Translation keys consist of two parts, a UI identifier and a label identifier. The UI identifier specifies what part of the user interface uses the data. It consists of the following elements

- Prefix (always the Percussion identifier, psx)
- UI Mode (identifies which general portion of the interface uses the data, for example, workflow, component, or slot)
- UI Context (identifies in detail the portion of the mode that uses the data; this element is optional)

For example, the UI identifier for Workflow TRANSITIONS table is psx.workflow.transition, while the UI identifier of the RXLOOKUP table is psx.keyword.

The label identifier specifies the label to use. Some tables only require the single column storing the label, but other may require an identifier column as well. In some cases, two identifiers are required. For example, the key for a Workflow Transition requires the WORKFLOWAPPID column as well as the TRANSITIONID column, while the key for a Role only requires the ROLENAME column.

You can use the Rhythmyx Language Tool to create Localized translations of display data from these tables.

The RXLOOKUP table is the most commonly used table in this list. It stores lookup data for droplists and similar controls used in Content Editors.

Several Rhythmyx resources have been internationalized to display localized data for these tables. These resources include:

| Application Name | Resources |
|----------------------------|---|
| sys_cmpUserCommunity | usercommunity |
| sys_caCommunityViews | All resources |
| sys_caContentSearch | All query resources |
| sys_caSites | casites_gen and casites_comm |
| sys_ceDependency | Depend, children and parents |
| sys_ceInlineSearch | resulpage, contenttypelookup |
| sys_ceSupport | contentstatus, variantlist, Variantlookup, lookup, variantlist |
| sys_cmpCaMenuNewContent | All resources |
| sys_cmpMenuViews | All resources |
| sys_i18nSupport | All resources |
| sys_commSupport | usercommunities, communityworkflows(2), communitynamelookup |
| sys_ca | All resources |
| sys_psxRelationshipSupport | workflowlookup, communityworkflowlookup |
| sys_rcSupport | relatedcontent, contentslotvariantlist, itemslotvariantlist, contenttypelookup, slotctypevariantlist, variantlist(2), variantlistwithslots(2), communitycontentlookup |

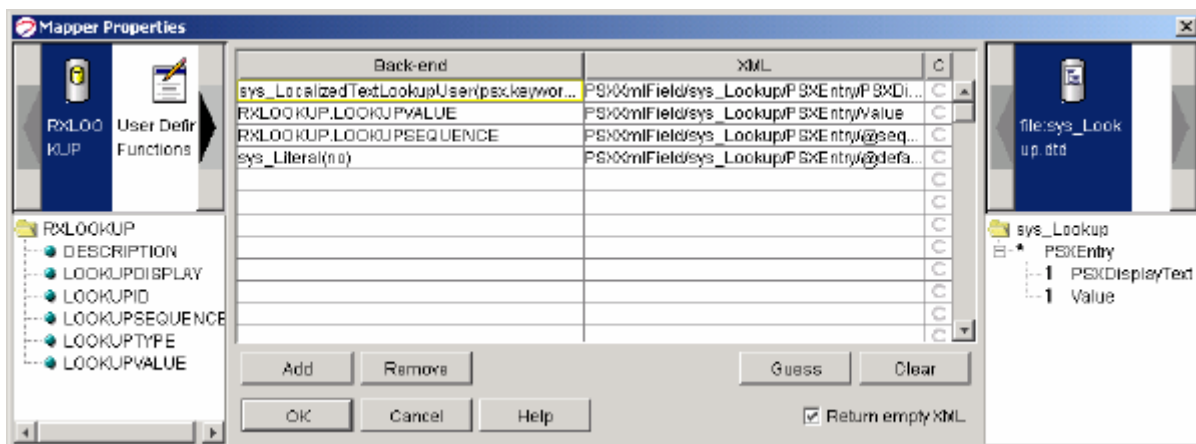
| Application Name | Resources |
|-------------------|---|
| sys_relatedSearch | resultpage, relatedsearch |
| sys_searchSupport | contentlookup, communitycontentlookup, stateslookup |
| sys_uiSupport | ActionList, ActionListChildren |
| sys_Variants | variants |

If your application links to these resources, it will already include localized data. For example, droplists that refer to the RXLOOKUP table usually use the Lookup resource in the application rx_ceSupport. This resource is already internationalized. If your Content Editor derives its data from this resource, it will present the user with localized data.

However, you will probably have to build applications that do not include these resources, particularly Content Assemblers, which are specific to each implementation. To retrieve localized data from internationalized tables for these applications, Rhythmyx includes two UDFs:

- `sys_LocalizedTextLookup`; this UDF requires you to specify how to derive the Locale used to lookup localized data. For example, you may use the Locale of the Content Item. Use this UDF when you cannot derive the Locale from the User Context, such as on Content Assemblers.
- `sys_LocalizedTextLookupUser`; this UDF derives the Locale from the User Context when looking up localized data. Use this UDF only when you can derive the Locale from the User Context, such as on Content Editors.

To use these UDF's map them to the DTD element associated with the localized data. The following graphic shows an example:



Both UDFs require you to specify the translation keys you want to use for the data. Key1 must specify the UI identifier while the remaining keys specify the table columns. In addition, the `sys_LocalizedTextLookup` UDF has a `LanguageString` parameter that specifies the location of the Locale for the translation keyword. When using the UDF on an assembler, for example, this parameter would probably be mapped to the Locale of the item.

Localizing the EditLive Text Editor

You can localize the interface of the `sys_EditLive` inline text editor control. For details see the topic "Adding Custom Menu and Toolbar Actions" in the Implementing Content Editors document or the Ephox EditLive! for Java Developer's Guide on the developer section of the Ephox Web Site (www.ephox.com (<http://www.ephox.com>)).

Implementing Internationalized Assembly

In addition to modifying Variant Assemblers for internationalization, you may need to modify dates based on Locale. Publishing some languages may also require modifying the output character set. Finally, you may want to automate the creation of links to localized versions of content.

Implementing Internationalization of Variants

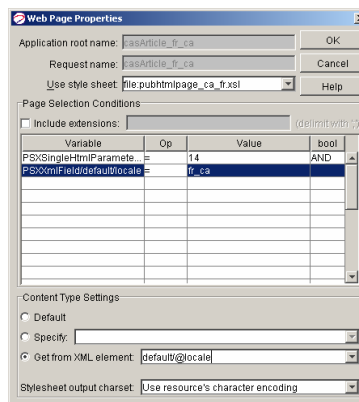
When developing Assembler applications for internationalization, you must be sure to include the CONTENTSTATUS.LOCALEID column in the Selector. You can choose to map this field to either an element or an attribute, but you cannot use the XML:lang attribute. Rhythmyx reserves this attribute to store the user's Locale.

Implementing Localized Variants

When implementing Localized Variants (unique Variants for each Locale), include all Locale-specific static text and image references normally. When defining the Web Page Properties, define the Variant ID and add a Boolean AND to a test for the Locale (or Locales) for which you defined the stylesheet:

| Variable | Operator | Value | Boolean |
|------------------------|----------|----------|---------|
| XMLElement/YourElement | = | LocaleID | None |

The following graphic shows a simple example:



If you use conditional processing in the stylesheet, you may need to add additional parameters testing for different LocaleIDs, linked by Boolean ORs.

Implementing Internationalized Variants

An internationalized Variant is a single Variant that produces different static text and images for different Locales. Two approaches are available for implementing internationalized Variants that include static text. You can use internatl conditional processing to display the correct text for the Locale, or you can use the Translation Bundle on the Rhythmyx server. To internationalize images, you must use conditional processing.

Consult an XSL reference for instructions about adding conditional processing to an XSL.

To use the Rhythmyx translation bundle, follow the instructions in “*Preparing XSL Files for Localization* (on page 9)” to add translation keys to your XSL, then follow the instructions for creating a translation XML (“*Generating TMX Files for Translation* (on page 18)”), translating the file, (“*Translating TMX Files* (on page 20)”), and uploading the translated file (“*Merging TMX Files with the Master TMX Document* (on page 20)”).

Implementing Internationalized Numbers and Date Formats

Formats for numbers and dates differ by Locale, and you often need to convert them from the default format in which they are stored in the database to the format appropriate to the Locale.

Numbers within content item text are part of the text string, which Rhythmyx does not translate automatically. The translator of the item must translate these numbers and dates.

Three UDFs are provided to convert date formats in system fields: `sys_ConvertDateLocale`, `sys_FormatDateToLocale`, and `sys_FormatDateToUserLocale`.

Using `sys_ConvertDateLocale`

You should usually use the `sys_ConvertDateLocale` UDF to convert dates. It is a generic UDF to convert dates from one format to another. It provides the flexibility to output different dates formats for different Locales. This UDF requires the following parameters:

| Parameter | Description |
|-------------------|--|
| DateToBeLocalized | Java Date object (most commonly from a backend column) or date string with a pattern allowed for the Locale of the system. |
| InputPattern | Input pattern used to parse the date. Must be a Java date format pattern. This parameter is only used if the value of the DateToBeLocalized parameter is a string value. If this parameter is not present, the Java.Medium format pattern is used, including both date and time. |
| InputLocaleString | String describing the input Locale of the date. Must use the Java Locale string format; see “ <i>Adding a New Locale</i> (on page 20)” for details. If you do not specify a Locale, the system’s default Locale is assumed. |

| Parameter | Description |
|--------------------|--|
| Output Pattern | The output patter for the date. Must be a Java date format pattern. If you do not specify a pattern, the Java.Medium format is used. |
| OutputLocaleString | String describing the output locale for the date. If this parameter is omitted, the system uses the user's Locale string. |

Using sys_FormatDateToLocale

This is a special version of the sys_ConvertDateLocale UDF that assumes that the input pattern and Locale are the system defaults. Use this UDF when you certain that the date uses the system's default date format, such as for system dates. This UDF requires the following parameters:

| Parameter | Description |
|--------------------|--|
| DateToBeLocalized | Java Date object (most commonly from a backend column) or date string with a pattern allowed for the Locale of the system. |
| Output Pattern | The output patter for the date. Must be a Java date format pattern. If you do not specify a pattern, the Java.Medium format is used. |
| OutputLocaleString | String describing the output locale for the date. If this parameter is omitted, the system uses the user's Locale string. |

Using sys_FormatDateToUserLocale

This UDF is a special version of the sys_FormatDateToLocale that takes the output Locale from the user's context. Use this UDF when you are certain that the date uses the system's default date format, such as a system date, and you are also certain that the output Locale is the user's Locale. This UDF requires the following parameters:

| Parameter | Description |
|-------------------|--|
| DateToBeLocalized | Java Date object (most commonly from a backend column) or date string with a pattern allowed for the Locale of the system. |
| Output Pattern | The output patter for the date. Must be a Java date format pattern. If you do not specify a pattern, the Java.Medium format is used. |

Changing Output Character Sets

The output character set defines the character set used in the pages Rhythmyx generates. The default output character set of Rhythmyx is UTF-8. The following additional character sets are available:

- ASCII
- ISO-8859-1
- ISO10646-UCS-2
- UTF-16
- UTF-16BE
- UTF-16LE

To change the output character set in an assembler application:

- 1 Start the Rhythmyx Workbench and open the application
- 2 In the Workbench Menu bar, choose *Tools > Options*.
Rhythmyx displays the Options dialog.
- 3 In the **XSL output encoding** field, choose the character set you want to use for the application from the drop list.
- 4 Click [OK].

Creating Automatic Links to Other Localized Versions of Content

You may want content to cross-link to other localized versions (for example, a Page in English would include cross-links to versions of the same Page in French, German, and Spanish). To create automatic links from one piece of content to other localized versions of the content, create an Automated Index (see “Implementing Automated Index Content Types” in the online Help or “Automated Index” in the printable document *Implementing Content Editors and Assemblers*).

The content query for this Automated Index will include the table PSX_RELATIONSHIPS. The ID of the content item could be either of two different columns in this table (PSX_RELATIONSHIPS.OWNERID or PSX_RELATIONSHIPS.DEPENDENTID). Because the GUI WHERE Table in the Selector does not provide the facilities for defining the precedence of selector statements, use manual SQL to select the data. Your SQL statement should include statements like the following:

```
(PSX_RELATIONSHIPS.OWNERID = CONTENTSTATUS.CONTENTID OR
PSX_RELATIONSHIPS.DEPENDENTID = CONTENTSTATUS.CONTENTID) AND
PSX_RELATIONSHIPS.CONFIG = TRANSLATION
```

(Note: The value of PSX_RELATIONSHIPS.CONFIG may be *Translation Mandatory* or a customized Relationship whose Category is *Translation*.)

More than likely, you will want to create your link on a snippet of text indicating the Locale of the content to which you want to link. In that case, use conditionals in your assembly XSL to select the correct text to display for the Locale of the item to which you are linking.

Localizing the Business User Interface

Implementing internationalized Workflow processes does not require any special work. While you may need to develop special processes for specific Locales, those processes do not need to change to accommodate different Locales.

The Business User interface, however, does require translation.

Localizing the Interface

To localize the interface, add a new language to Rhythmyx. Because Rhythmyx uses the Java definition of language by Locale, you can add multiple variants of each language to Rhythmyx. For example, you can add French as spoken and used in France and French as spoken and used in Canada.

When adding a new language, you translate text inherent in Rhythmyx including:

- Text in Rhythmyx CMS tables;
- Labels in Content Explorer (other than the //Sites and //Folders roots, which cannot be translated);
- Displayable references in stylesheets;
- Displayable references in extension and system resources (System resources produce labels and links within the Rhythmyx interface [for example, the Insert, Edit, and View Dependents labels in the Action Menu], as well as error codes for exits provided by Percussion Software that have been localized.)

Rhythmyx uses Translation Memory Exchange (TMX), an XML-standard method of defining translatable data. For details about TMX, see <http://www.lisa.org/tmx/tmx.htm>. The Rhythmyx server installation includes a command-line tool, the Rhythmyx Language Tool, that you can use to add new Locale's to Rhythmyx, create TMX files for translation, and upload translated TMX files to Rhythmyx.

While CMS table text and Content Editor labels are available automatically, you must modify XSL files if they contain any of the following:

- Static text
- Alternative text
- Locale-specific image references or cascading stylesheets

You may also want to localize cascading stylesheets.

Any JavaScript files included in XSL must also be localized if they include alert or prompt messages should also be modified for translation.

You may also want to use Locale-specific images in your content editor, especially if the images include any text.

Note that when you add a new Locale to Rhythmyx, the Rhythmyx Language Tool only copies system resources (Locale-specific resources shipped as part of Rhythmyx) to the new Locale. Any resources specific to an implementation, such as graphics or JavaScript files, must be copied to Locale-specific folders manually.

Rhythmyx Language Tool

The Rhythmyx Language Tool is a command-line interface that allows you to:

- add a new Locale to Rhythmyx;
- create TMX files for translation;
- merge translated TMX files with the master internal TMX file; and
- enable and Disable Locales.

The Rhythmyx Language Tool is installed with the Rhythmyx server, and is stored in the `rxlt` directory of the Rhythmyx server directory.

Run the language tool from a command line. To run the Rhythmyx Language Tool, change to the `rxlt` directory, and run the `rxlt` batch file or shell script.

When the Rhythmyx Language Tool starts, it will display the available actions:

- 1 Generate TMX File
- 2 Merge Translated TMX File with server TMS File
- 3 Add a Locale
- 4 Enable or disable a Locale
- 5 Edit display name of Locale
- 6 Exit

Adding a New Locale

Rhythmyx uses a variation of the Java definition of Locale, a combination of two, two-letter codes to define each Locale. For example, the Java definition for US English is `en_US`, while `de_DE_EURO` is the definition for a specific version of German spoken in Europe. Rhythmyx uses a modification of this system that is commonly used in XML, in which all the characters are lowercase and are separated by hyphens rather than underscores. In this system, the definition for US English is `en-us`, while the definition for the European form of German is `de-de`.

The first two-letter code in the Locale definition specifies the language. These codes use the ISO 639-1 two-letter definition for languages. For a list of these codes, see <http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt>. To order a copy of this ISO standard, see <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=22109&ICS1=1&ICS2=140&ICS3=20>. (NOTE: Three-letter definitions are also available for these languages, but Rhythmyx does not recognize these codes.)

The second two-letter code in the Locale Definition specifies the country. These codes use the ISO 3166 two-letter definitions for languages. For a list of these codes, see <http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html>.

For example, if you wanted to add Canadian French to your system, the Locale definition would be `fr-ca`. The `fr` indicates that the language is French. The `ca` indicates that the country is Canada.

To add a new Locale to Rhythmyx:

- 1 Start the Rhythmyx Language Tool.
- 2 Enter 3 (Add Language).

- 3** Enter the Locale string in the form <Language>-<Country>. For example, add Canadian French to your system, you would enter *fr-ca*.
- 4** Enter a Display name for the Locale; for example, *Canadian French*.
- 5** Enter a Description for the Locale; for example, *French as spoken and used in Canada*. (NOTE: The Description is optional).
- 6** Press the <Enter> key.

The Rhythmyx Language Tool adds the new Locale to your system.

When you add a new Locale, the Rhythmyx Language Tool makes copies of all Locale-specific resources from the default Locale directories to directories for the new Local. For example, if you add the new Locale *de-de*:

- The Rhythmyx Language Tool copies the `templates.css` file from the directory `sys_resources/css/en-us` to the directory `rx_resources/css/de-de`. You can modify this file to make the colors, fonts, and other elements specific to the Locale.
- The Rhythmyx Language Tool copies the file `globalErrorMessage.js` from the directory `sys_resources/js/en-us` to the directory `rx_resources/js/de-de`. You will need to translate this file for the new Locale.
- The Rhythmyx Language Tool copies all images from the directory `sys_resources/images/en-us` to the directory `rx_resources/images/de-de`. You will need to modify these images for the new Locale.

Note that when you add a new Locale, it is always disabled, which gives you time to generate and translate the TMX file before enabling the Locale.

Editing the Display Name and Description of a Locale

To edit the display name and description of a Locale:

- 1** Start the Rhythmyx Language Tool.
- 2** Enter 5 (Edit display name of Locale).
- 3** Enter the Locale string of the Locale whose display name and description you want to modify.
- 4** Enter the new display name for the Locale. Note that you must enter the entire text. To skip this step, press the <Enter> key.
- 5** Enter the new description of the Locale. Note that you must enter the entire text. To skip this step, press the <Enter> key.
- 6** Press <Enter> to continue.

The Rhythmyx Language Tool displays the current list of supported Locales, and the Available Options menu.

Generating TMX Files for Translation

Before generating the TMX file, you should prepare all *XSL file* (see "Preparing XSL Files for Localization" on page 20)s and *JavaScript* (see "Localizing JavaScript Files" on page 26) files for localization, and *localize all images* (see "Using Locale-specific Images" on page 26).

To generate a TMX file for translation:

- 1 Start the Rhythmyx Language Tool.
- 2 Enter 1 (Generate TMX Resources).
- 3 Rhythmyx displays a list of interface sections to translate:
 1. CMS Tables
 2. XSL Stylesheets
 3. Content Editors
 4. Extension Resources
 - a) All
- 4 Enter the sections you want to include. You can enter more than one section. Separate sections with commas. NOTE: If you enter "A", the Rhythmyx Language Tool ignores any additional characters you enter.
- 5 Press the <Enter> key.
- 6 Specify the path and name of the output file. The default file and location is rxlt/tobetranlated.tmx.
- 7 The Rhythmyx Language Tool asks you whether you want to include only missing keywords from the master resource in the generated TMX file. Enter "yes" (default) to include only missing keywords. Enter "no" to include all keywords in the TMX file. Usually, "yes" is the correct option.

Rhythmyx generates the TMX file in the location you specified.

Translating TMX Files

Rhythmyx TMX files use only a subset of the TMX DTD.

Each TMX file starts with a header that is generated by the Rhythmyx Language Tool. Do not modify the data in the header. For example, if you attempt to add a language by adding it to the header of a TMX file instead of registering it through RLT, Rhythmyx will not recognize the new language.

The header includes one <prop> element for every Locale currently supported in the CMS implementation.

```
<prop type="supportedlanguage">de-de</prop>
<prop type="supportedlanguage">en-us</prop>
<prop type="supportedlanguage">fr-ca</prop>
<prop type="supportedlanguage">it-it</prop>
<prop type="supportedlanguage">ja-jp</prop>
```

The Rhythmyx Language Tool generates translation keywords automatically. Each translation keyword is defined by a Translation Unit (<tu>) element. The Translation Unit Identifier (tuid) attribute specifies the translation keyword.

```
<tu tuid="psx.ce.action@Check-out">
```

Each Translation Unit element must include one <prop> child. (NOTE: The TMX DTD allows multiple <prop> children, but Rhythmyx permits only one.) The <prop> child of the <tu> element specifies the section name, which defines the CMS area in which the keyword is used.

Each <tu> element may include one <note> child. This element provides information to all translators about translating the keyword. (NOTE: The TMX DTD allows multiple <note> children, but Rhythmyx permits only one.)

Each <tu> element has a Translation Unit Variant (<tuv>) child for each Locale defined in the header. Each <tuv> element has a <seg> child. Enter the translation for the keyword for the Locale in the <seg> element. The <tuv> element can also include a <note> child to provide information about translating the keyword to a specific Locale. No XML comments are allowed in TMX documents, however. XML comments in a TMX document will not be uploaded with the document.

Merging TMX Files with the Master TMX Document

To merge a translated TMX file with the Master TMX document:

- 1 Start the Rhythmyx Language Tool.
- 2 Enter 2 (Merge translated TMX file with master).
- 3 Specify the path and name of the TMX file you want to merge with the Master TMX document. For example, `rxlt/tobetranslated.tmx`.
- 4 Press the <Enter> key.

Updating TMX Files

You can update the translations in the server TMX file. For example, if you change terminology, you would update the TMX file to use the new terminology. You can even change the terms in the default Locale of the server TMX file. For example, you may want to use “Checkout” instead of the default “Check-out”. You can modify the translation of the keyword to the new value.

To update a TMX File, generate a TMX file for the interface elements you want to retranslate, retranslate the file, and merge the retranslated file with the server TMX file.

Enabling and Disabling Locales

When you initial add a Locale to Rhythmyx, it is disabled so you can translate it and merge it with the Master TMX document. To make the Locale available in your system, you must enable it.

Before enabling the Locale, complete all translations. The translation should include all localized *XSL files* (see "[Preparing XSL Files for Localization](#)" on page 26) and *JavaScript files* (see "[Localizing JavaScript Files](#)" on page 27). All *Images* (see "[Using Locale-specific Images](#)" on page 27) should also be localized before you enable the Locale.

To enable a Locale:

- 1 Start the Rhythmyx Language Tool.

- 2 Enter 4 (Enable/disable a Language).
- 3 Enter the Locale string of the Locale you want to enable. For details about the Locale string, see “*Adding a New Locale* (on page 23)”.
- 4 Press the <Enter> key.

Use the same procedure to disable a Locale.

Removing Stale Keywords

Keywords may become stale or out of date if you no longer use them. For example, if you delete a Workflow, the Transitions no longer exist and the Transition labels are no longer useful. You can leave these keywords in your server TMX file without harm to your system, but the server TMX file will be unnecessarily large. Therefore, you may want to remove these stale keywords.

The Rhythmyx Language Tool does not provide an option for removing stale keywords, but you can use the following procedure:

- 1 Run the Rhythmyx Language Tool and generate a TMX file. Choose “A” to include all interface sections in the file.
- 2 When the RLT asks whether you want to include only missing keywords, enter “no”.
- 3 Generate the TMX file and save it locally to the Rhythmyx Language Tool as ResourceBundle.tmx.
- 4 Create a backup copy of the server TMX file (`./rxconfig/I18n/ResourceBundle.tmx`) and store it in a safe location.
- 5 Replace the server copy of ResourceBundle.tmx with the new copy saved in the Rhythmyx Language Tool directory.

Preparing XSL Files for Localization

You do not have to modify all XSL files for localization. You only have to modify XSL files if they include the following elements:

- Static text
- Alt text
- Locale-specific images or cascading stylesheets.

To modify an XSL file for translation:

- 1 Add the internationalization namespace to the XSL file:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:psxi18n="urn:www.percussion.com/i18n" exclude-result-
prefixes="psxi18n">
```

- 2 Import the `sys_I18nUtils.xsl` file:

```
<xsl:import href="file:sys_resources/stylesheets/sys_I18nUtils.xsl"/>
```

- 3 Add a global variable called ‘lang’ just after all the imports.

```
<xsl:variable name="lang" select="//@xml:lang"/>
```

If you import the XSL file into any other XSL files, the main XSL file in the set (the XSL into which the all others are imported) must have the variable definition. The `select` attribute in the variable definition assumes that the root element of the XML document has the `xml:lang` attribute. Note that the `xml:lang` attribute is reserved for the user's Locale string. If you want to add a different Locale (for example, the Locale of the item), you must use a different name.

- 4 To localize static text, replace the text you want to localize with a template call. The template call should have two parts: the first is a unique key for the text and the other is a text string. The recommended format is:

```
applicationname.xslfilename@texttobetranlated
```

where

`applicationname` is the name of the Rhythmyx application;

`xslfilename` is the name of the XSL file; and

`texttobetranlated` is the actual text that will be localized.

For example, suppose you wanted to localize the following:

```
<tr>
  <td class="datacell1font">Search Results</td>
</tr>
```

The modified code would look like this:

```
<tr>
  <td class="datacell1font">
    <xsl:call-template name="getLocaleString">
      <xsl:with-param name="key"
select="'psx.applicationname.xslfilename@Search Results'"/>
      <xsl:with-param name="lang" select="$lang"/>
    </xsl:call-template>
  </td>
</tr>
```

- 5 Use the same technique to localize alt text that you use to localize static text. The recommended format is:

```
applicationname.xslfilename.alt@AltText
```

Where

`applicationname` is the name of the Rhythmyx application;

`xslfilename` is the name of the XSL file; and

`alt` indicates that the template is for alt text; and

`AltText` is the text that will be localized.

For example:

```

<xsl:attribute name="alt">
<xsl:call-template name="getLocaleString">
<xsl:with-param name="key"
select="'psx.sys_cmpCaSearchBox.searchbox.alt@Calendar Pop-up'"/>
<xsl:with-param name="lang" select="$lang"/>
</xsl:call-template>
```



```
</xsl:attribute>
</img>
```

6 If you develop Locale-specific images, place them in Locale-specific subdirectories of the `rx_resources/images` directory. For example:

```
../rx_resources/images/fr-ca
../rx_resources/images/de-de
../rx_resources/images/ja-jp
```

Use the following code to define the `src` attribute of localized images:

```
src="concat(' ../rx_resources/images/', $lang, '/imagename.gif) "
```

For example:

```

```

7 To make all of the keys available to the Rhythmyx Language Tool, list them all as children of the `psxi18n:lookupkeys` element at the end of the XSL file. For example:

```
<psxi18n:lookupkeys>
<key name="psx.applicationname.xslfilename@Search Results">Search
Results text for Search Box.</key>
<key name="sys_cmpCaSearchBox.searchbox.alt@Calendar Pop-up">Alt text
for calendar pop-up.</key>
</psxi18n:lookupkeys>
```

The value of the `name` attribute is the translation keyword recovered by the Rhythmyx Language Tool, while the value of each key is the note to the translators for the translation unit.

8 When processing, the XSL file needs to know the user's Locale. The Locale is user's session object `sys_lang` variable. In the mapper of the application associated with the XSL, add `xml:lang` as an attribute of the root element of the result document.

| | |
|--|-----------------------------------|
| PSXUserContext/User/SessionObject/sys_lang | PSXXmlField/RootElement/@xml:lang |
|--|-----------------------------------|

Add the same mapping to any application that imports this XSL.

Localizing JavaScript Files

Localize JavaScript files if they include alert messages or prompts.

To localize JavaScript files, replace the text of alert messages and prompts with a Localized Message function call and a keyword:

```
alert(LocalizedMessage "alert_message_keyword");
```

Localized error messages are stored in the

`rx_resources/js/Locale/globalErrorMessages.js` JavaScript file. Each Locale has a unique version of this JavaScript file located in a Locale-specific subdirectory of the `rx_resources/js` directory. For example, if your system include US English, German, and Canadian French Locales, you would have the following `globalErrorMessages.js` files:

```
rx_resources/js/en-us/globalErrorMessages.js
rx_resources/js/de-de/globalErrorMessages.js
rx_resources/js/fr-ca/globalErrorMessages.js
```

Keyword entries are added to the `Psx_MessageMap_Local` variable. The format for keyword entries in these files is:

```
alert_message_keyword: "Locale specific translation of the keyword.",
```

NOTE: All entries MUST end with a comma.

For example, if you had the following JavaScript function:

```
function search_submit()
{
    if(document.formx.itemx.value=="")
    {
        alert("Search name is a required field");
    }
    return false;
}
document.formx.submit();
}
```

You modify the alert message:

```
function search_submit()
{
    if(document.formx.itemx.value=="")
    {
        alert(LocalizedMessage("search_name_required"));
    }
    return false;
}
document.formx.submit();
}
```

Next, add an entry for the keyword to the `PsxMessageMap_Locale` variable of the `rx_resources/js/en-us/globalErrorMessages.js` JavaScript file.

```
var PsxMessageMap_Locale ={
    search_name_required: " Search name is a required field.",
    save_search_prompt:  "Please enter a name for this search.\nNote: search
will be overwritten if name already exists.",
    .....
    .....
};
```

Add the keyword entry to the `globalErrorMessages.js` JavaScript file for each Locale. Assuming your implementation included the three Locales specified above, you would also have to add an entry to the `rx_resources/js/de-de/globalErrorMessages.js` and `rx_resources/js/fr-ca/globalErrorMessages.js` JavaScript files.

Any XSL files that use localized JavaScript files must include both the system (`sys_resources`) and local (`rx_resources`) `globalErrorMessages.js` JavaScript files and the Global Variable `lang` mapped to the `xml:lang` attribute of the root element of the XML. See “*Preparing XSL Files for Localization* (on page 27)”, Step 8 for instructions about mapping the `lang` Global Variable. To add the `globalErrorMessages.js` JavaScript files, include the following script calls in the XSL file:

```
<script language="javascript"
src="../../sys_resources/js/globalErrorMessages.js"></script>
<script language="javascript"
src="{concat('../../rx_resources/js/', $lang, '/globalErrorMessages.js')}"><
/script>
```

Using Locale-specific Images

Images may need to be localized, particularly if they include text. Rhythmyx does not include tools for localizing images themselves. Use appropriate graphic tools to develop Locale-specific images. The images for each Locale should be stored in a Locale-specific subdirectory of the `rx_resources/images` directory. For example, if your implementation includes US English, Canadian French, and Japanese, you would have the following subdirectories:

```
rx_resources/images/en-us
rx_resources/images/fr-ca
rx_resources/images/ja-jp
```

When coding XSL files, use the following code to define the `src` attribute of localized images:

```
src="concat('../rx_resources/images/', $lang, '/imagename.gif) "
```

For example:

```

```

Defining Translation Properties

By default, when you create a new Translation Content Item, Rhythmyx adds it to the same Workflow and Community as its owner. To specify a different Community and Workflow, you must define a configuration that specifies a different Workflow and Community for each new Content Item of a specific Content Type in a specific Locale. Use the Auto Translation Configuration Editor to define these configurations.

| Auto Translations | | | | |
|-------------------|-------------------|-----------------|-----------------------------------|---------------|
| New Configuration | | | | |
| | Content Type (ID) | Locale | Community (ID) | Workflow (ID) |
| ✗ | Article (1) | Canadian French | North American Marketing (1002) | Article (1) |
| ✗ | Article (1) | Mexican Spanish | Central American Marketing (1003) | Article (1) |
| ✗ | Article (1) | US English | North American Marketing (1002) | Article (1) |
| ✗ | Brief (3) | Canadian French | North American Marketing (1002) | Article (1) |
| ✗ | Brief (3) | Mexican Spanish | Central American Marketing (1003) | Article (1) |
| ✗ | Brief (3) | US English | North American Marketing (1002) | Article (1) |

NOTE: If you want to allow users to specify the Workflow and Community when creating a Translation Content Item, you must create a new Workflow Action that provides an alternate interface where the user can specify these values rather than using the default interface.

Auto Translation Edit Configuration Page

Use the Auto Translation Edit Configuration page to define Community and Workflow to assign new Content Items when generated in a specific Locale. To access the Auto Translation Edit Configuration page,

- on the Auto Translation Configuration Editor, click [New Configuration](#).
- on the Auto Translation Configuration Editor, click the name of the Content Type in the row with the Locale for which you want to edit the Auto Translation Configuration.

| Edit Configuration | |
|---|------------------------------|
| Content Type : Locale | |
| Community: | Central American Marketing ▾ |
| Content Type: | Article Word ▾ |
| Workflow: | Article ▾ |
| Locale: | Mexican Spanish ▾ |
| <input type="button" value="Save"/> <input type="button" value="Cancel"/> | |

Field Descriptions

Content Type Mandatory (read-only when editing a configuration). The Content Type for which you are defining the configuration.

Locale Mandatory (read-only when editing a configuration). The Locale for which you are defining the configuration. Available options include all enabled Locales.

Community Mandatory. Drop List. The Community assigned to the Auto Translation configuration. Options include all Communities with which at least one Role and one Content Type has been associated.

Workflow Mandatory. Drop List. The Workflow assigned to the Auto Translation configuration.

Adding a Translation Configuration

To add a Translation Configuration:

- 1 On the *Auto Translation Configuration Editor* (see "[Defining Translation Properties](#)" on page 31), click [New Configuration](#).
Rhythmyx displays the *Auto Translation Edit Configuration page* (on page 32).
- 2 Select the **Content Type** and **Locale** for which you are defining the configuration.
- 3 Select the **Workflow** and **Community** to which you want to assign new Translation Content Items of the specified Content Type generated in the specified Locale.
- 4 Click the **[Save]** button to save your configuration.


Editing a Translation Configuration

You can change the Community and Workflow of the Auto Translation Configuration. To change the configuration:

- 1 On the *Auto Translation Configuration Editor* (see "[Defining Translation Properties](#)" on page 32), click the name of the Content Type in the row of the Locale for which you want to change the configuration.
Rhythmyx displays the *Auto Translation Edit Configuration page* (on page 32) with the current data for the configuration you selected.
- 2 You can change the **Workflow** and **Community** for the Configuration. The Content Type and Locale are read-only.
- 3 Click the **[Save]** button to save your changes.

Deleting a Translation Configuration

If you delete an Auto Translation configuration, new Translation Content Items will revert to the same Community and Workflow as the Owner that created them.

To delete an Auto Translation Configuration, on the *Auto Translation Configuration Editor* (see "[Defining Translation Properties](#)" on page 32), click the delete button  in the row of the Content Type and Locale for which you want to delete the configuration.

Implementing Automated Translation

When working with a Strong Dependency model, you usually want to generate Translation Content Items automatically at a specific point in the Workflow. Before implementing Automated Translation, the following Rhythmyx elements must be in place:

- Translation Content Types
- Locales to be translated (must also be enabled)
- Workflows
- Communities (at least Roles and Content Types must be associated with each Community to make it eligible to use in Auto Translation.
- Translation Configurations

To implement Auto Translation, Add the `sys_createTranslations` Workflow Actions to the Workflow Transitions where you want to generate the Translation Content Items.

Example Auto Translation Implementation

To demonstrate implementation of Auto Translation, let us begin with a system with in which Canadian French (fr-ca) and Mexican Spanish (es-mx) locales have been enabled. The Translations will use the system's Article Content Type.

The system includes a Central American Marketing Community and a North American Marketing Community.

| | | Communities New Community |
|-------------------------------------|----------------------------|------------------------------|
| Name (ID) | Description | |
| ✗ Central American Marketing (1003) | Central American Marketing | |
| ✗ Default (10) | Default Community | |
| ✗ North American Marketing (1002) | North American Marketing | |

The Central American Marketing Community includes a Spanish Translation Workflow:

| | | Workflows for Central American Marketing(1003) Community Add Workflow |
|---------------------------|------------------------------|--|
| Name (ID) | Description | |
| ✗ Article (1) | Basic Article Workflow | |
| ✗ Images (3) | Image Workflow | |
| ✗ Index (2) | Index Pages and Briefs | |
| ✗ Spanish Translation (4) | Spanish Translation Workflow | |

The North American Marketing Community includes a French Translation Workflow:

| Workflows for North American Marketing(1002) Community Add Workflow | | |
|---|------------------------|-----------------------------------|
| | Name (ID) | Description |
| ✗ | Article (1) | Basic Article Workflow |
| ✗ | French Translation (5) | Translate Content Items to French |
| ✗ | Images (3) | Image Workflow |
| ✗ | Index (2) | Index Pages and Briefs |

Example Auto Translation Configurations

When generating Article Content Items in the Mexican Spanish (es-mx) Locale, we want the Content Items to go into the Central American Marketing Community and into the Spanish Translation Workflow:

| Edit Configuration | |
|---|------------------------------|
| Content Type : Locale | Article : Mexican Spanish |
| Community: | Central American Marketing ▾ |
| Content Type: | Article |
| Workflow: | Spanish Translation ▾ |
| Locale: | Mexican Spanish |
| <input type="button" value="Save"/> <input type="button" value="Cancel"/> | |

When generating Article Content Items in the Canadian French (fr-ca) Locale, we want the Content Items to go into the North American Marketing Community and the into the French Translation Workflow:

| Edit Configuration | |
|---|----------------------------|
| Content Type : Locale | Article : Canadian French |
| Community: | North American Marketing ▾ |
| Content Type: | Article |
| Workflow: | French Translation ▾ |
| Locale: | Canadian French |
| <input type="button" value="Save"/> <input type="button" value="Cancel"/> | |

Example Transition for Auto Translation

Best practice when generating Translation Content Items automatically is to spin off the Translation Content Item as late in the Workflow as possible, so most of the work on the original is complete. Therefore, we will add the `sys_createTranslations` Workflow Action to the Approve Transition from the QA State to to Public in the system's Article Workflow:

| Workflows > Article > QA > Approve | |
|-------------------------------------|--|
| ID: | 5 |
| *Label: | Approve |
| Description: | |
| *Trigger: | Approve |
| From-State: | QA (3) |
| To-State: | Public |
| Approval Type: | Specified Number |
| *Approvals Required: | 1 (Required if Approval Type is set to "Specified Number") |
| Comment: | Optional |
| Default Transition: | N |
| Workflow Action: | sys_createTranslations |
| Transition Role: | - All roles - |
| <input type="button" value="Save"/> | |

Example Auto Translation Content Item

With the implementation complete, let us take a Content Item through the Workflow. We begin with a sample Content Item:

*** System Title:** Lorem ipsum dolor sit amet

*** Title:** Lorem ipsum dolor sit amet

*** Start Date:** 2004-02-15

Expiration Date: 2004-05-31

Reminder Date: 2004-01-31

Keywords: Lorem ipsum dolor sit amet




Description: Lorem ipsum dolor sit amet

Callout:

veniam, quis nostrud exercitation ullam corper suscipit looboris nisi ut auquip ex ea commodo consequat. Duis autem velem iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel willum lunombro dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Design Code

When this Content Item Transitions from the QA State to the Public State, two Translation Content Items are generated automatically:

| Content Path://Views/All Content/All | | | |
|---|------------|----------------------------|---------------------|
| △ Name | State | Community Name | Workflow Name |
|  [es-mx] Copy (2) of Lorem i... | Translate | Central American Marketing | Spanish Translation |
|  [fr-ca] Copy (1) of Lorem ip... | Translate | North American Marketing | French Translation |
|  Lorem ipsum dolor sit amet | Quick Edit | Default | Article |

Configuring the sys_createTranslations Workflow Action

By default, the sys_createTranslations Workflow Action creates a new Content Item of the same Content Type as the Owner in each enabled Locale in the system, with a Translation - Mandatory Relationship between the Owner and each Dependent.

You can change the configuration of the Action to create a different Relationship or to skip certain Locales. These properties are controlled by the sys_createTranslation.properties file (<Rhythmyxroot>\rxconfig\I18N\sys_createTranslations.properties).

To change the Relationship between the Owner and the Translation Dependent, change the value of the sys_relationshiptype property to the name of the Relationship you want to use instead of Translation - Mandatory.

To skip automatic creation of Translation Content Items in a specific Locale, to the sys_excludelocales property. Use commas or semicolons to separate values.

Implementing Internationalized Publishing

Each internationalization model of Publishing requires a minor modification to implementation.

Implementing Language-centric Publishing Models

To implement a language-centric publishing model, you will need to create a Site for each Locale, with associated Locale-specific Editions and Content Lists.

The Content List application query in this model includes a test for the Locale string of the Locale for which the Content List is created:

```
CONTENTSTATUS.LOCALE = 'de-de'
```

The registrations of the Site, Editions, and Content Lists do not require any special features to implement this model.

Implementing Content-centric Publishing Models

To implement a content-centric publishing model, you only need to define the Location Schemes for the Contexts to include the Locale in the path or the name of each output. For example, the graphic below shows the registration of a Location Scheme that would produce the following path:

```
../articles/Locale/art_contentid.htm
```

The screenshot shows the 'Edit Location Scheme' window for a Location Scheme named 'Article(1)'. The configuration fields are as follows:

- Name: Article
- Description: Article scheme
- Generator: Java/global/percussion/contentassembler/sys_casConcatAssemblyLocation
- Content Type: Article
- Variant Type: Published HTML Page

Below the configuration fields is a table titled 'Location Scheme Parameters' with the following data:

| Name(id) | Type | Sequence | Value |
|----------|---------------|----------|-------------------------|
| root | String | 0 | /articles/ |
| Locale | BackendColumn | 1 | CONTENTSTATUS.LOCALEID |
| Middle | String | 2 | /art_ |
| middle | BackendColumn | 3 | Contentstatus.contentid |
| file | String | 4 | .html |

For example,

```
../articles/en-us/art_1345.htm
```

Full-text Search in Globalized Environments

The Convera Full-text Search Engine shipped with Rhythmyx includes only the search libraries for the default Locale, EN-US. If your implementation includes other Locales, the Full-text Search engine requires additional search libraries for those Locales to index and return results for those Locales.

If your search engine does not include the search libraries required for a specific Locale, the first time you add a Content Item to that Locale, Rhythmyx will return an error similar to the following (which assumes the additional Locale fr-ca):

```
fr-ca not supported, using default of en
```

Rhythmyx returns this error in the server window, and in the server log.

Additional search libraries are available from Percussion Software. Contact your Percussion Software Sales Representative for details about obtaining these additional libraries.

If a user enters text in an unsupported Locale, the full-text search engine indexes it in the default Locale, en-us. Thus, if a user enters French text that include the word *chat* (cat), for example, this word would be expanded according to the English meaning of the string, resulting in matches such as "talk" or "converse".

Index

A

- Adding a New Locale • 19, 23, 27
- Adding a Translation Configuration • 33
- Assembly Engine • 8, 18
- Auto Translation Edit Configuration Page • 32, 33
- Automated Index • 21

C

- Changing Output Character Sets • 21
- Character sets • 5, 21
- Configuring the sys_createTranslations Workflow Action • 39
- Content Engine • 7, 13
- Content Relationships • 9
- Content-centric Publishing • 11, 40
- Creating Automatic Links to Other Localized Versions of Content • 21

D

- Database character set • 5
- Date Formats • 19
- Defining the Content Relationship Model for Translations • 9
- Defining Translation Properties • 32, 33, 34
- Deleting a Translation Configuration • 34
- Determining System Requirements • 5

E

- Editing a Translation Configuration • 33
- Editing the Display Name and Description of a Locale • 24
- Enabling and Disabling Locales • 26
- Example Auto Translation Configurations • 36
- Example Auto Translation Content Item • 38
- Example Auto Translation Implementation • 35
- Example Transition for Auto Translation • 37

F

- Full-text Search in Globalized Environments • 41

G

- Generating TMX Files for Translation • 19, 25

I

- Images • 31
- Implementing Automated Translation • 35
- Implementing Content-centric Publishing Models • 40
- Implementing Internationalization • 13
- Implementing Internationalization of Variants • 18
- Implementing Internationalized Assembly • 18
- Implementing Internationalized Numbers and Date Formats • 19
- Implementing Internationalized Publishing • 40
- Implementing Internationalized Variants • 19
- Implementing Language-centric Publishing Models • 40
- Implementing Localized Variants • 18
- Internationalization Support on the Rhythmyx Server • 14
- Introduction to Internationalization and Localization in Rhythmyx • 3

J

- JavaScript • 29

L

- Language-centric Publishing • 11, 40
- Locale
 - Java definition • 3
 - Rhythmyx definition • 23
- Localizing JavaScript Files • 25, 26, 29
- Localizing the Business User Interface • 22
- Localizing the EditLive Text Editor • 17
- Localizing the Interface • 22

M

- Merging TMX Files with the Master TMX Document • 19, 26
- Modeling Assembly for Internationalization • 8
- Modeling Internationalization of a Rhythmyx Content Management System • 7
- Modeling Internationalized Workflow Processes • 9
- Modeling Internationalized Workflows • 9
- Modeling Publishing • 11

N

Number Formats • 19

P

Preparing XSL Files for Localization • 8, 19, 25, 26, 27, 30

Publishing Engine • 11, 40

R

Removing Stale Keywords • 27

Rhythmyx Language Tool • 22, 23, 25, 26, 27

S

Spinning Off Translation Versions of Content Items • 10

String dependency • 9, 10

T

Translating TMX Files • 19, 25

Translation keys • 14

U

Updating TMX Files • 26

User-defined Functions • 14, 19, 20

Using Locale-specific Images • 25, 26, 31

Using sys_ConvertDateLocale • 19

Using sys_FormatDateToLocale • 20

Using sys_FormatDateToUserLocale • 20

V

Variants • 8, 18, 19

W

Weak dependency • 9

Workflow Engine • 9, 22

Workflow Processes in Strong Dependency Models • 10