Rhythmyx

Implementing the Relationship Engine

6.5.2

Printed on 10 May 2007

Copyright and Licensing Statement

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.

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

CHAPTER 1

Implementing Relationships in Rhythmyx

In Rhythmyx, a Relationship is a logical association between two Rhythmyx objects. Rhythmyx functions that use Relationships include:

- Active Assembly (the association between a Content Item and its related content is a Relationship)
- Folders (the association between a folder and a Content Item contained in it is a Relationship)
- Promotable Versions (the association between the original Content Item and the new Version is a Relationship; in this case, the Relationship has special processing, called an Effect, that moves the original Content Item to an Archive State when the new Version becomes Public)
- Globalization (when you create a new copy of a Content Item for Translation, Rhythmyx creates a Relationship between the original and the Translation Copy; in this case, the user has the option of specifying a Relationship that allows the two Content Items to go Public independent of one another (non-Mandatory Relationship), or whether they must go Public together (Mandatory Relationship).

Most commonly, the objects involved in the Relationship are Content Items, but note above that Relationships are also used to implement the folder functionality in the Content Explorer user interface and can be extended to incorporate other Rhythmyx objects as well.

All Rhythmyx Relationships have the following properties:

- Rhythmyx only recognizes one-to-one Relationships. One-to-many, many-to-one, and many-to-many Relationships are not valid. An object may be related to several other objects, but each Relationship is a unique entity.
- In each Relationship, one object owns the relationship (and is referred to as the owner). The other object is the dependent in the Relationship. A dependent in one Relationship, however, may be the owner in another Relationship. In that case, the dependent in the second Relationship is a descendant of the owner in the first Relationship. The owner in the first Relationship is the ancestor of the dependent in the second Relationship.
- A Relationship exists as long as the owner exists and the Relationship itself is not actively removed (such as being manually removed or automatically destroyed due to system processing). If an object is deleted, all of its Relationships are also deleted.

Components of Rhythmyx Relationships

The following components comprise Rhythmyx Relationships.

- Properties (mandatory)
- ***Cloning options*** (see "Cloning" on page 4) (optional)
- ***Effects*** (on page 5) (optional)

Properties

Relationship properties define general information about the Relationship, such as its name, its sort order, and whether it is used in Active Assembly.

An option available for all properties of Relationships (both default properties and user-defined properties) is to Lock the Relationship. Locking prevents processing in extensions from overriding the specified value of the property. If a property is not specified as locked, extensions can use local values to override specified values.

Percussion Software provides the following default properties for all Relationships:

Property	Values	Default Value	Locked	Description
rs_useownerrevision	yes no	yes	Yes	Defines whether to use the owner revision as part of the owner locator.
rs_usedependentrevision	yes no	no	Yes	Defines whether to use the dependent revision as part of the dependent locator.
rs_useserverid	yes no	yes	No	Specifies the user Rhythmyx uses when executing Effects. If the value of this property is yes (default), Rhythmyx uses RXSERVER. If the value of this property is no, Rhythmyx uses the current user. Rhythmyx throws an exception if the current context does not specify a user.

You can also add custom properties (User Properties) for Relationships.

Cloning

Cloning options specify whether you can clone the Relationship when cloning the Content Item. You can either enable or disable cloning for a specific Relationship.

If you enable cloning, you can specify whether to create a shallow clone or a deep clone. A shallow clone duplicates only the Relationships to the dependents (and owners) of the Content Item you are cloning. A shallow clone is typically used for simple copies of Content Items, such as the creation of a new ***promotable Version*** (see "Promotable Relationship Processing" on page 13) of an Item.

A deep clone duplicates all of the Relationships associated with a Content Item, including Relationships to descendants (dependents of dependents) and ancestors (owners of owners); for more information, Dependency Relationship Processing. A common use of this option is to create Translation Copies of Content Items in globalized environments, when you need to be sure that all the content associated with an item is translated. In both cases, you can specify conditions to determine when that type of cloning is permitted.

You can enable either shallow or deep cloning, or enable both. If you enable only one form of cloning for a Relationship, Rhythmyx always uses that type of cloning when processing the Relationship. If you enable both types of cloning, you must specify conditional processing to determine which type of cloning will occur when processing the Relationship. Rhythmyx executes the first type of cloning whose conditions evaluate as TRUE. Thus, since shallow cloning is processed before deep cloning, if a situation where the conditions for both types of cloning evaluate as TRUE, Rhythmyx would create a shallow clone of the Relationship, since the shallow cloning option comes before the deep cloning option.

Finally, you can specify whether to allow user-defined cloning properties to override default properties with the same name in the Relationship.

Effects

Effects are Rhythmyx extensions available only for Relationship processing. Use Effects to provide processing for Relationships after they have been created. (Use Exits to provide processing when creating Effects). An Effect defines a series of instructions executed when the Effect is triggered. The Conditions assigned to the Effect determine when it will be triggered. A common Condition assigned to an Effect is to trigger it only when a Content Item is Transitioning into a specific State, or when a Transition cannot occur. For example, in a globalized environment, you might want to inform Translators when a Translation Copy of a Content Item has been created. In that case, you would assign the Effect sys_Notify to the Relationship, and add conditions that specified it was triggered when the command equaled *relate*/*create*. The graphic below shows an example:

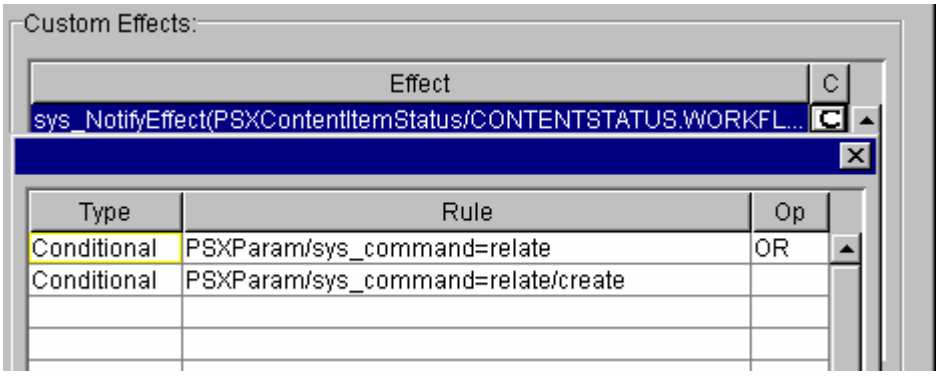

Figure 1: Example Using Conditions to Define Trigger for an Effect

Example of Relationships in Action

To understand how Relationships work in Rhythmyx, lets look at some examples. Let us begin with a simple HTML page that consists of some text and a graphic:  . This page consists of two Content Items, the system's Article Content Item that contains the text and the system's Image Content Item that is used to manage the Image file. The two Content Items are associated through an Active Assembly Relationship.

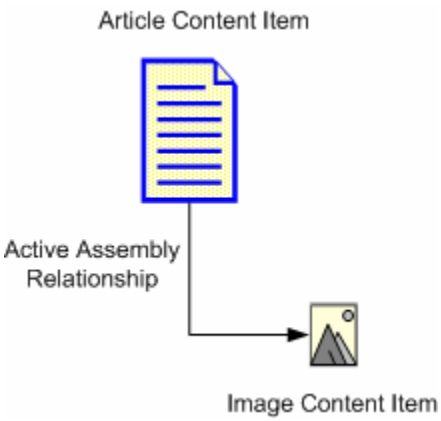

Figure 2: Active Assembly Relationship

The Active Assembly Relationship is a very basic Relationship. It simply points to a Content Item to insert into a Slot in a specific Variant of a Content Item. At assembly, the individual Content Items will be formatted, then the HTML page will be formatted with a reference to the image.

Now, let us suppose we want to ensure that the Article cannot go Public unless the associated graphic is also ready to go Public. The Active Assembly Relationship does not meet our needs because it does not put any constraints on the two Content Items. Each can go Public independent of the other. The Article will be Published if it is Public, but depending on the Authorization Type of the Content List, the Image may or may not be Published.

To ensure that the Article cannot go Public unless the associated Image is ready to go Public, we use the Active Assembly – Mandatory Relationship.

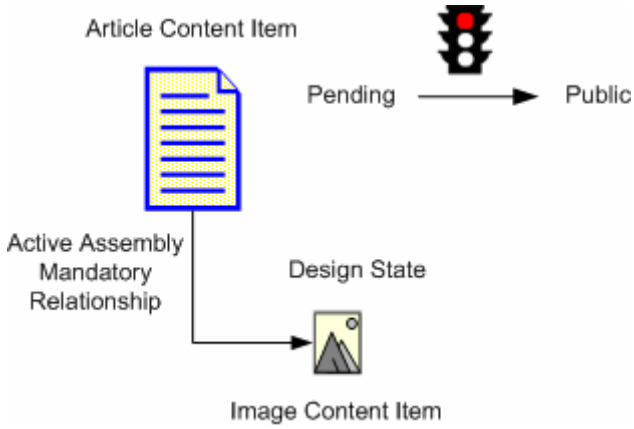

Figure 3: Active Assembly - Mandatory Relationship with Dependent not in a Public State

The Active Assembly – Mandatory Relationship includes the Effect sys_PublishMandatory:

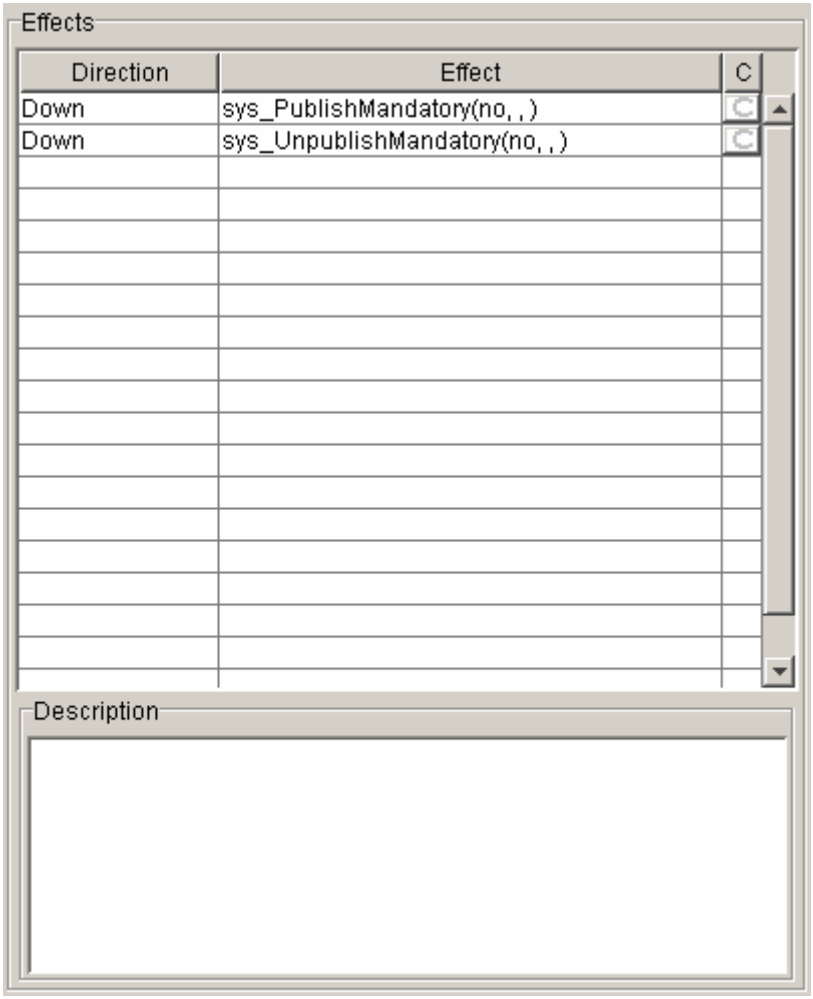

Figure 4: Configuration of the sys_PublishMandatory Relationship

This Effect prevents a Content Item from going Public if the associated Content Item in the Relationship is not also Public. The Direction configured for the Effect determines whether the Effect will be triggered by the Owner or the Dependent in the Relationship. In this case, the Direction is Down, which means the Effect is triggered by the Owner and checks whether the Dependent is Public. Thus, in our case, if the Image Content Item is not Public, we cannot Transition the Article to Public. The Article will wait in the Pending State until the Image Content Item (as well as all other Dependents in Mandatory Relationships) has entered the pending State. When all of these Dependents enter the Pending State, we will be able to Transition the Article to Public.

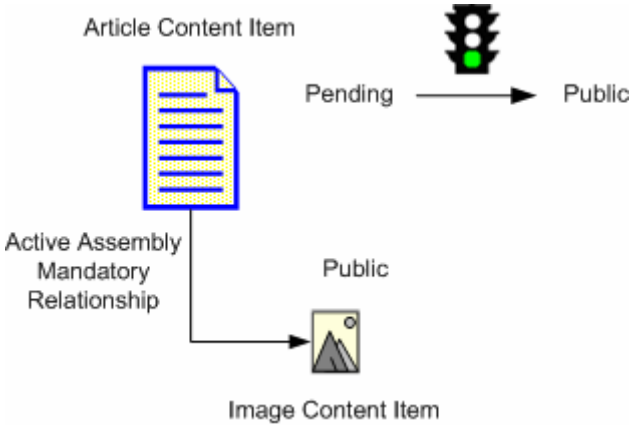

Figure 5: Active Assembly - Mandatory Relationship with Dependent in a Public State

Forcing Items to Public

Sometimes, you may want to use one action to move multiple Content Items (for example, a system's Article Content Item and its associated Image Content Items and other Dependent Content Items) to Public.

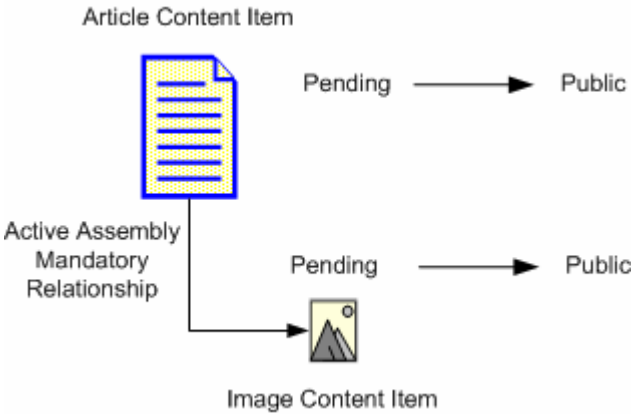

Figure 6: Forcing a Dependent Content Item to Public

Whether one action Transitions multiple Content Items is controlled by the forceTransition parameter of the sys_PublishMandatory Effect. If the value of this parameter is no, then the Dependents in the Relationship will not be Transitioned with their Owners. If the value of this parameter is yes, then all Dependents that use that Relationship Type will be forced to Transition along with the Owner.

You must also specify the name of the Transition that you want Rhythmyx to use to Transition the Dependent Content Item. In the case of the Active Assembly – Mandatory Relationship, since the Direction is Down, you must specify a value for the dependentTransitionName parameter. (NOTE: If you do not specify a Transition, Rhythmyx uses the Default Transition from the State to make the Transition.) The following graphic shows an example reconfiguration:

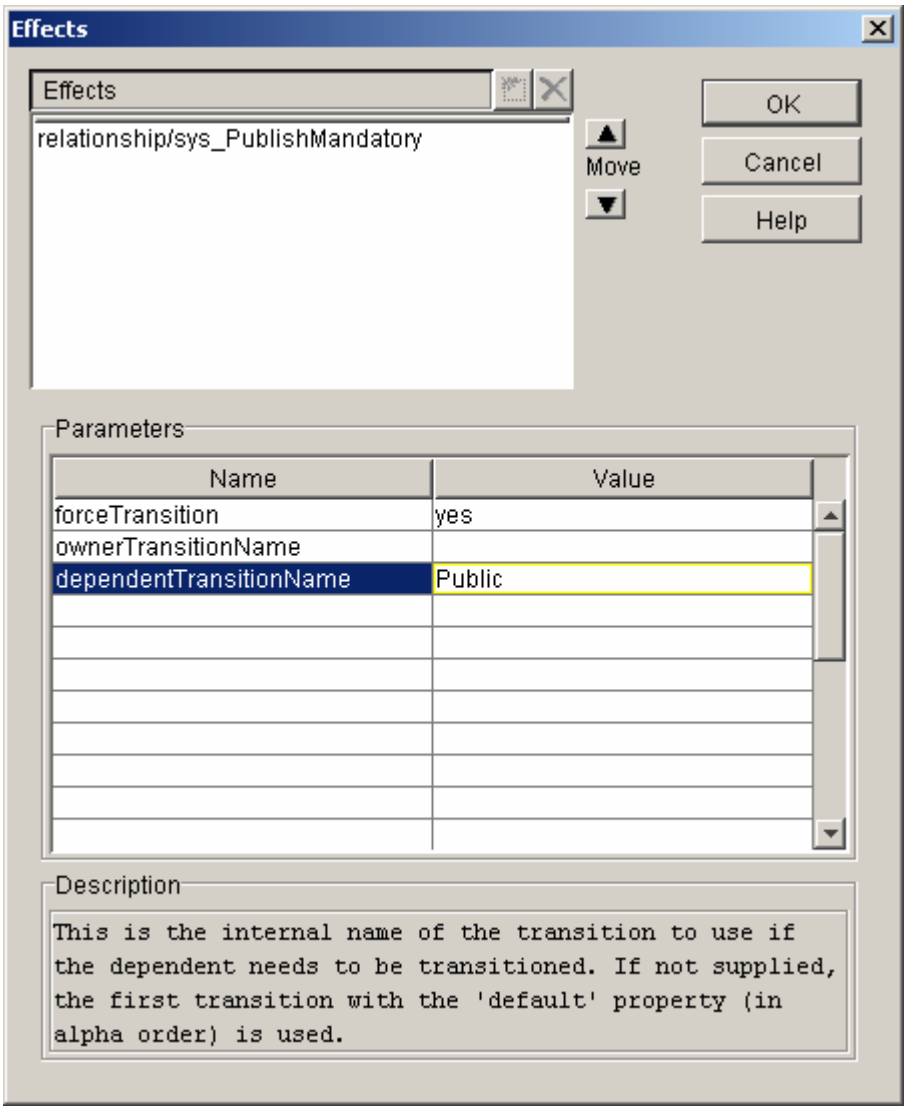

Figure 7: Reconfiguration of the sys_PublishMandatory Relationship to a force a Dependent Content Item to Public

With this configuration, if all Dependent Content Items have reached a Pending State (the State prior to the Public State), transitioning the Article Content Item to Public will force all of the Dependent Content Items in the Pending State to Public as well.

Advanced Example: Translations

Let us now assume that we work in a internationalized environment, and we want to translate our Content Item into French and Japanese. Let us also assume the following:

- We do not need a different graphic when we translate to French, but we do need a different graphic when we translate to Japanese.
- Our system includes an Article Content Type
- The English Article Content Item can go Public regardless of the current State of the Japanese Translation.
- The English Article Content Item can only go Public if the French Content Item is ready to go Public as well.

To meet these differing objectives, we will use the Translation – Mandatory Relationship to create the French translation, but we will use the Translation Relationship to create the Japanese translation.

Note that to accomplish these objectives, we also need a slight modification to the default configuration of the Cloning properties of the Active Assembly Relationship. The default Cloning properties of the Active Assembly Relationship call for deep cloning (cloning of both the Relationship and the associated Content Item) if cloning is triggered by a Relationship in the Translation Category.

For the purpose of this exercise, we will assume that this condition has been removed. Instead, we will assume conditions based on the Locale of the Translation Content Item:

- If the Translation goes to the French Locale, Rhythmyx will shallow clone the Active Assembly Relationship.
- If the Translation goes to the Japanese Locale, Rhythmyx will deep clone the Relationship.

Using the Translation - Mandatory Relationship to Create the French Translation Content Item

To create the French Translation, we use the *Create* > *Translation* - *Mandatory* action in Content Explorer. This action clones our English Article Content Item, and links the English Article (Owner) to the clone (Dependent) using a Translation – Mandatory Relationship. The Relationship links the English Content Item to its French Dependent so you can use Impact Analysis to track it. The Relationship also prevents the creation of more than one Translation Dependent for any Locale in the system.

Like the Active Assembly – Mandatory Relationship, configuration of the Translation – Mandatory Relationship includes the sys_PublishMandatory Effect. The Direction specified for the Effect in this case is Up, which prevents the English Content Item from going Public unless the French content Item is already public.

As noted above, we have changed the cloning properties of the Active Assembly Relationship to create a shallow clone in the French Locale. Both the English Content Item and its French Dependent will use the same graphic.

The following graphic illustrates the French Translation:

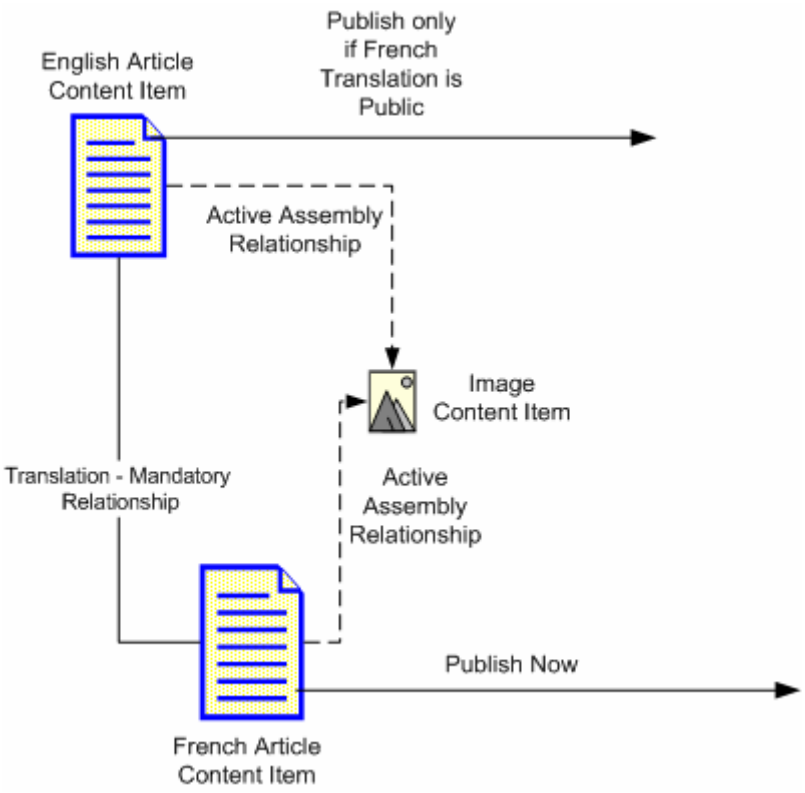

Figure 8: Creating the French Translation Content Item

Using the Translation Relationship to Create the Japanese Translation Content Item

To create the Japanese Translation, we use the Create – Translation action in Content Explorer. As in the French example, this action clones the English Article Content Item and links the English Article to the clone, but this time uses Translation Relationship. Unlike the Translation – Mandatory Relationship, this Relationship does not include the sys_PublishMandatory Effect. Therefore, the two Content Items can go Public regardless of the State of the other Content Item in the Relationship.

As noted above, we have changed the cloning Properties of the Active Assembly Relationship to create a deep clone in the Japanese Locale. Thus, when Rhythmyx clones the Active Assembly Relationship from the English Content Item, to its graphic, it also clones the Image Content Item. This clone of the Active Assembly Relationship points to the cloned image Content Item.

The following graphic illustrates the Japanese Translation:

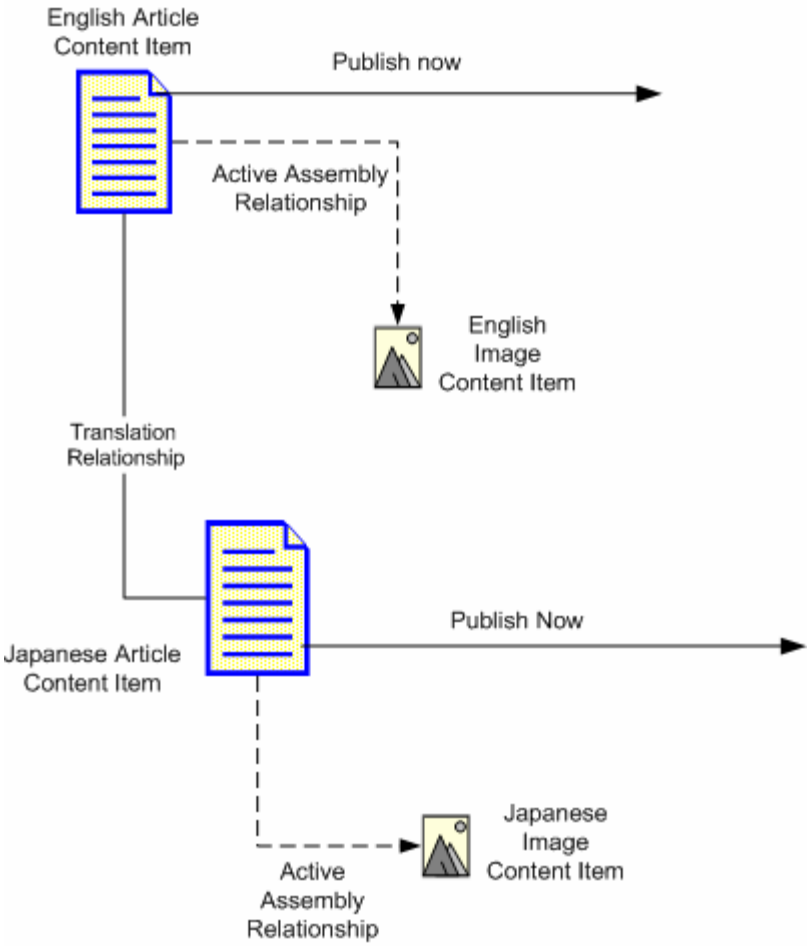

Figure 9: Creating the Japanese Translation Content Item

Relationship Processing

Relationships are created in two ways. Users may add Relationships explicitly through their actions. For example, a user that creates an Active Assembly association is creating a Relationship. Similarly, a user that assigns a Content Item to a folder in Content Explorer is also creating a Relationship. Rhythmyx can also create Relationships automatically. A Workflow Action, for example, might create a Relationship by creating a new Translation Copy of a Content Item.

Rhythmyx uses three tables to manage and store Relationships. The PSX_RXCONFIGURATIONS table stores configurations for Relationship Types. The Relationship records themselves are stored in the PSX_RELATIONSHIPS table. The PSX_RELATIONSHIPPROPERTIES table stores additional attributes used to process Relationships.

Several Rhythmyx subsystems (such as Workflow or Content Editors) can process Relationships, but all processing is generic, independent of the subsystem in which the processing occurs. The properties of a Relationship can determine when and how it is processed, but the majority of the processing is defined by the Effects associated with the Relationship, which define the processing that occurs, and by the conditions that trigger those effects. Effects may be triggered at the following times:

- When the Relationship is created or destroyed;
- When a Content Item is checked in or checked out;
- When a Content Item is Transitioned from one State to another;
- When a Content Item is cloned.

The points when an Effect may be triggered are called *execution contexts*. Each Effect must specify the execution context for which it runs. In the ***example Effect*** (on page 64), the execution context is Transitions, as defined by the following code:

Promotable Relationship Processing

A Promotable Relationship is a Relationship between a Content Item (Owner) and a clone (Dependent) of the item in which the clone will supersede the original when the clone becomes Public. The clone is typically called a new Version of the owner.

The sys_Promote Effect is used to implement Promotable Relationships processing. When the Dependent Content Item enters a Public State the first time, the sys_Promote Effect replaces the Owner with the Dependent. To execute the replacement:

- The Owner is Transitioned, using either the Transition specified in the transitionName parameter of the sys_Promote Effect or the default Transition from the Public State. The Transition used should move the original Owner to an Archive State.
- Updates all Relationships in which the Owner in the Promotable Relationship was specified as the Dependent to specify the newly-promoted Dependent in the Promotable Relationships as the Dependent.

- Removes all Clonable Relationships from the Owner in the Promotable Relationships.
- Updates all other Relationships that specified the Owner in the Promotable Relationships as the Owner to specify the newly-promoted Dependent in the Promotable Relationship as the Owner.

To illustrate how Promotable Versions work, let us examine the following sequence.

Let us begin with a Content Item that is currently Public. In general, Promotable Versions are most useful when you want to make a significant change to a Content Item that is already Public, such as major revision of the text of the Content Item. For minor revisions, such as correcting misspelled words, you would use the Quick Edit Transition to make the Content Item editable briefly while you make the correction. If the Content Item has not yet become Public, you would make the changes directly to the Item itself rather than creating a new Promotable Version of the Item.

For our purposes, our Public Content Item is Item 742 in the graphic below. Item 742 was created by copying Content Item 507 (Item 507 is the New Copy Owner of Item 742; Item 742 is the New Copy Dependent of Item 507) . In addition, another copy of Item 742 was created (Item 823; Item 742 is the New Copy Owner of Item 823; Item 823 is the New Copy Dependent of Item 742) for other reasons. In addition, we have a Translation of Item 742 into French.

Item 742 is the Dependent in an Active Assembly Relationship, with Content Item 498 as it's Owner. Item 742 itself owns an Image in an Active Assembly Relationship.

The following graphic illustrates the current Relationships of Content Item 742:

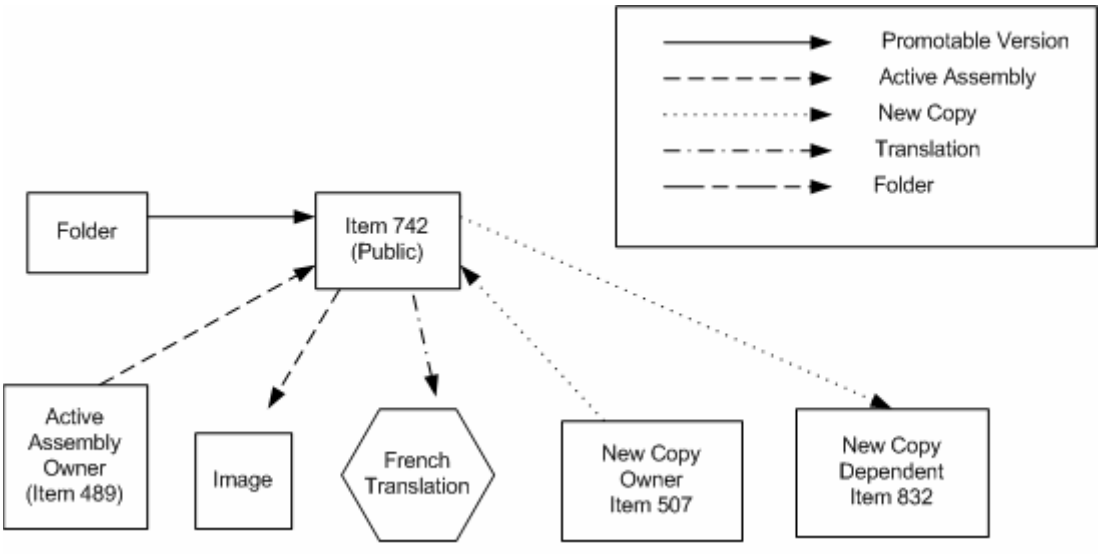

Figure 10: Initial Relationships of Item 742

When we create a new Promotable Version of Item 742, a clone of Item 742 (Item 914) is created, and the Active Assembly Relationship between Item 742 and its Active Assembly Dependent image is also cloned, as illustrated in the following graphic:

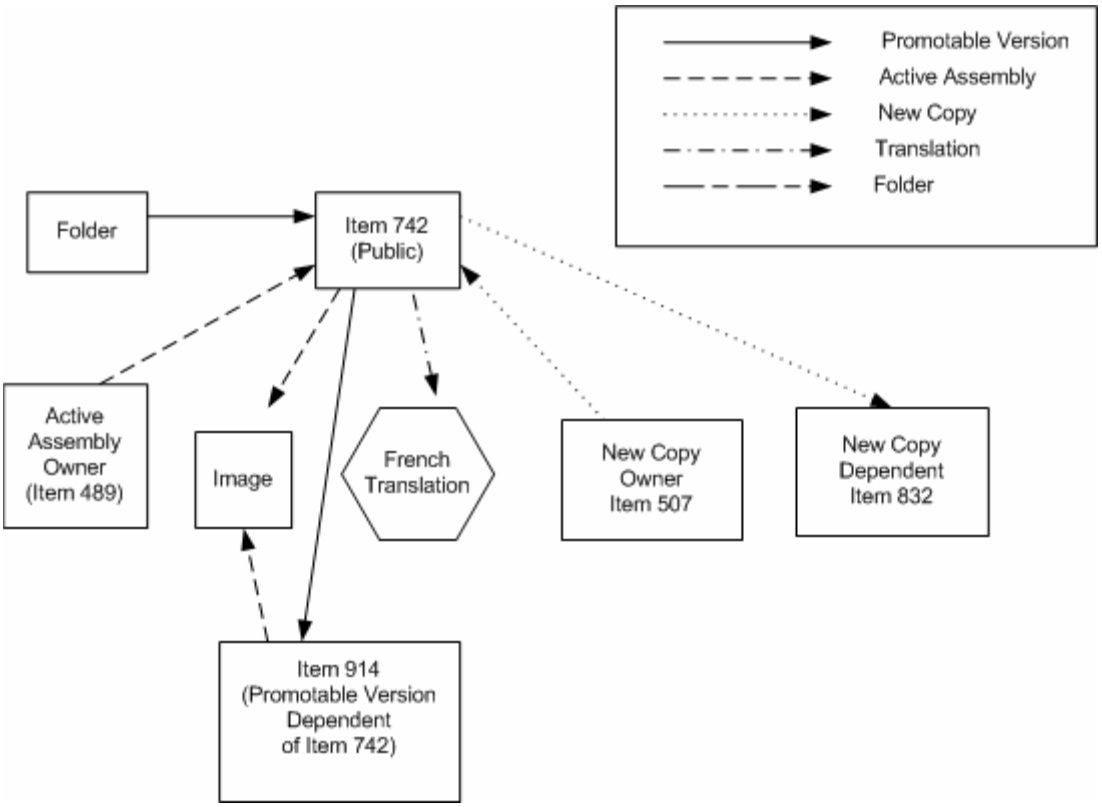

Figure 11: Relationships following the Creation of the Promotable Version Content Item (914)

When the Promotable Version Content Item (914) goes Public, Item 742 is Transitioned to the Archive State. The Active Assembly Relationship in which Item 742 was the Dependent is now re-pointed to make Item 914 the Dependent. The clonable Active Assembly Relationship between Item 742 and its Dependent image is deleted. The Active Assembly Relationship between Item 914 and the Image (which was a clone of the Relationship between Item 742 and the image) is not changed. When Item 914 is Published, the image will be included.

The remaining Relationships to Item 742 are re-pointed to Item 914. Thus, in the New Copy Relationship to Item 507, in which Item 742 was the Dependent, Item 914 is now the Dependent. At the same time, the New Copy Relationship to Item 832 and the Translation Relationship to the French Translation, in both of which Item 742 was the Owner, Item 914 is now the Owner.

The following graphic illustrates the state of all Relationships after all Promotable Version processing is complete.

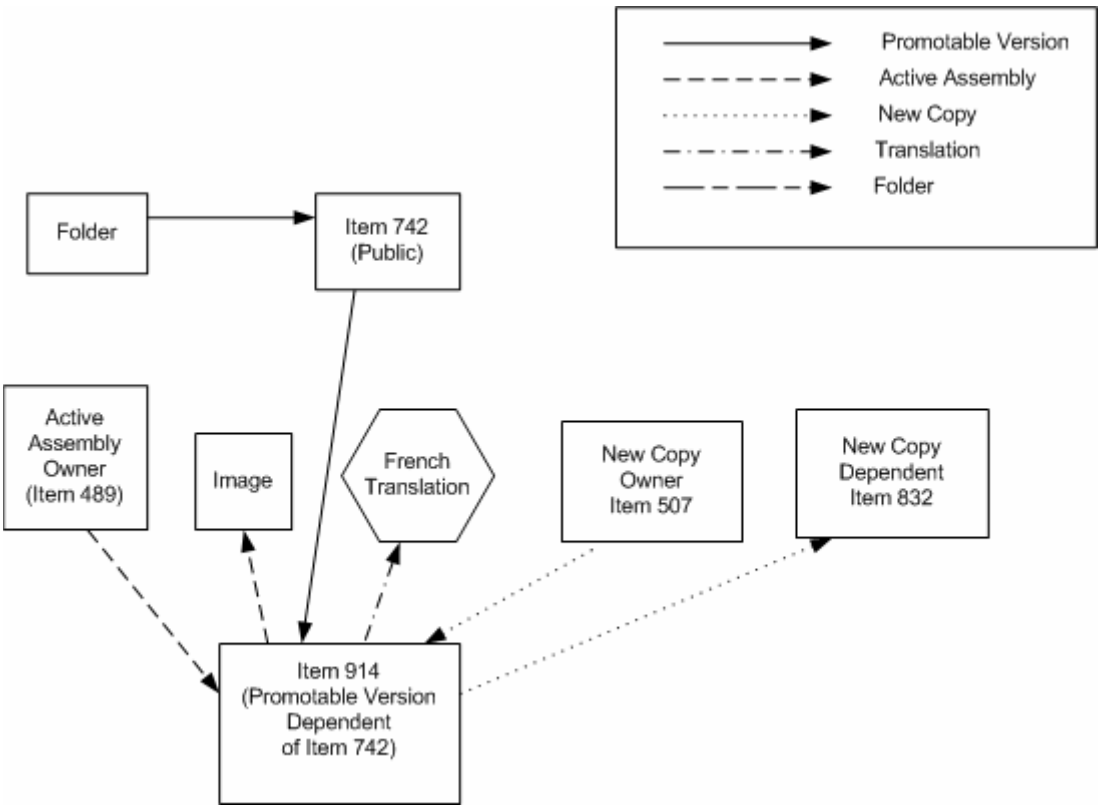

Figure 12: Relationships following the promotion of Content Item 914 to Public

NOTE: While you can create multiple Versions (promotable clones) of a Content Item, only the first clone to be Transitioned to Public will supersede the Owner. Any other promotable clones that are Transitioned to Public will be made Public as if they were not in a Promotable Relationship (none of the other Relationships will be re-pointed and the promoted Content Item will not supersede the currently Public item. Note that a superseded Owner that is returned to Public acts in the same manner as a competing clone; it becomes Public as if it were not in a Promotable Relationship, none of the other Relationships are re-pointed and it does not replace the currently Public Content Item.

Mandatory Relationships and Workflows

If you choose to implement a mandatory Relationship, you must pay special attention to your Workflows.

Since mandatory Relationships require that the Owner and Dependent in the Relationship both go Public together, a poorly designed Workflow can allow content to become trapped, unable to progress to Public. Best Practice when designing Workflows that might be used by a mandatory Relationship is to include "pending" State immediately prior to the Public State. This State acts as a marshalling area for Content Items for which all work is effectively complete and which are ready to go Public, but which must wait for Dependent Content Items to reach the same State before they can make the final Transition to Public.

Another issue to consider is whether you want to force Transitions on Content Items. You might want to force Transitions in two cases:

- Several Content Items are waiting in a "pending" State. When you Transition one of them, you want to Transition all of them.
- A Content Item is in a "pending" State, but it's Dependent is not there yet. You want both to go Public regardless of the current State of the Dependent Content Item.

To facilitate forced Transitions, you need to specify one of the Transitions from a State as the Default Transition. To make a Transition the Default Transition, choose Y from the Default Transition drop list on the when defining the Transition on the Edit Transition page in Content Explorer. (Note: If you specify more than one Transition as the Default Transition from a State, Rhythmyx uses the first Transition from the State in alphabetical order among those specified as a Default Transition.)

You will need an Effect to implement your forced Transition. For example of an Effect that forces a Transition, see the ***sys_PublishMandatory Effect*** (see "sys_PublishMandatory" on page 96).

CHAPTER 2

Relationship Dialogs

The Rhythmyx Workbench provides two dialogs for maintaining Relationship Type:

- The ***Relationship wizard*** (see page 20) is used to create new Relationships
- The ***Relationship Type editor*** (see page 22) is used to complete a Relationship Type or to change its configuration

Another dialog, the Relationship Effects Execution Contexts dialog, is used to define the processing context in which the Relationship Effect processing will occur.

New Relationship Type Wizard

The New Relationship Type wizard allows you to create a new Relationship type object that appears in the System Design view; however, the Relationship is not usable until you complete the Relationship Type editor.

To access the New Relationship Type wizard:

- In System Design view, right-click on the Relationship Types folder and choose *New* > *Relationship Type*.
- From the Menu bar choose *File* > *New* > *Other*. In the Select a Wizard dialog, choose *Relationship Type.*

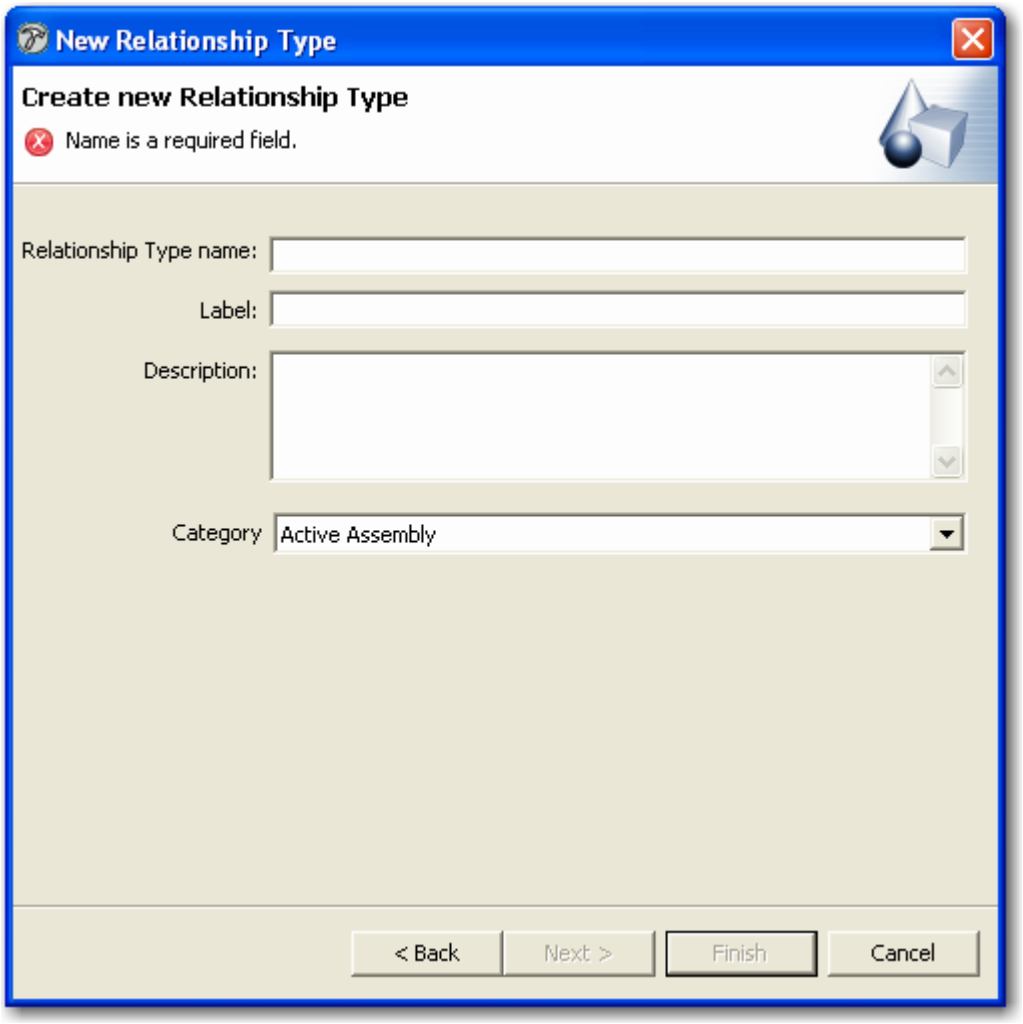

Figure 13: New Relationship Type Wizard

Field Definitions

Relationship type name - Required. System name for Relationship type.

Label - Required. Name displayed to users for Relationship type.

Description - Description of Relationship type.

Category - The category of the Relationship. Options are *Active Assembly*, *New Copy*, *Folder*, *Promotable Version*, *Translation*.

Display Name	**Internal Name**
Active Assembly	rs_activeassembly
	NOTE: Relationships in this Category require the following User Properties: sys_slotid, sys_variantid, and sys_sortrank.
New Copy	rs_copy
Folder	rs_folder
Promotable Version	rs_version
Translation	rs_translation

Relationship Type Editor

Although the Relationship Type object is completed in the New Relationship Type wizard, the Relationship Type editor includes the fields required to make it functional. Users also use the Relationship Type editor to edit existing Relationship Types.

To access the Relationship Type editor:

- After completing the ***New Relationship Type wizard*** (see page 20), click [**Finish**].
- Right-click on the Relationship Type object in any view that displays it and select *Open.*
- Double-click on the Relationship Type object in any view that displays it.

Using the Relationship Type editor, you can:

- ***Add properties to the Relationship Type*** (see "Adding Properties to the Relationship Type" on page 35)
- ***Edit properties of the Relationship Type*** (see "Editing Properties of a Relationship Type" on page 36)

The Relationship Type editor includes four tabs:

- ***General tab*** (see "Relationship Type Editor, General Tab" on page 23)
- ***Properties tab*** (see "Relationship Type Editor, Properties Tab" on page 25)
- ***Cloning tab*** (see "Relationship Type Editor, Cloning Tab" on page 27)
- ***Effects tab*** (see "Relationship Type Editor, Effects Tab" on page 29)

Relationship Type Editor, General Tab

When a Relationship type opens in the Relationship type Editor, the General tab displays the values of common Relationship properties for this Relationship type.

Use this tab to:

- ***add general properties of the Relationship type*** (see "Adding Properties to the Relationship Type" on page 35).
- ***edit general properties of the Relationship type*** (see "Editing Properties of a Relationship Type" on page 36).

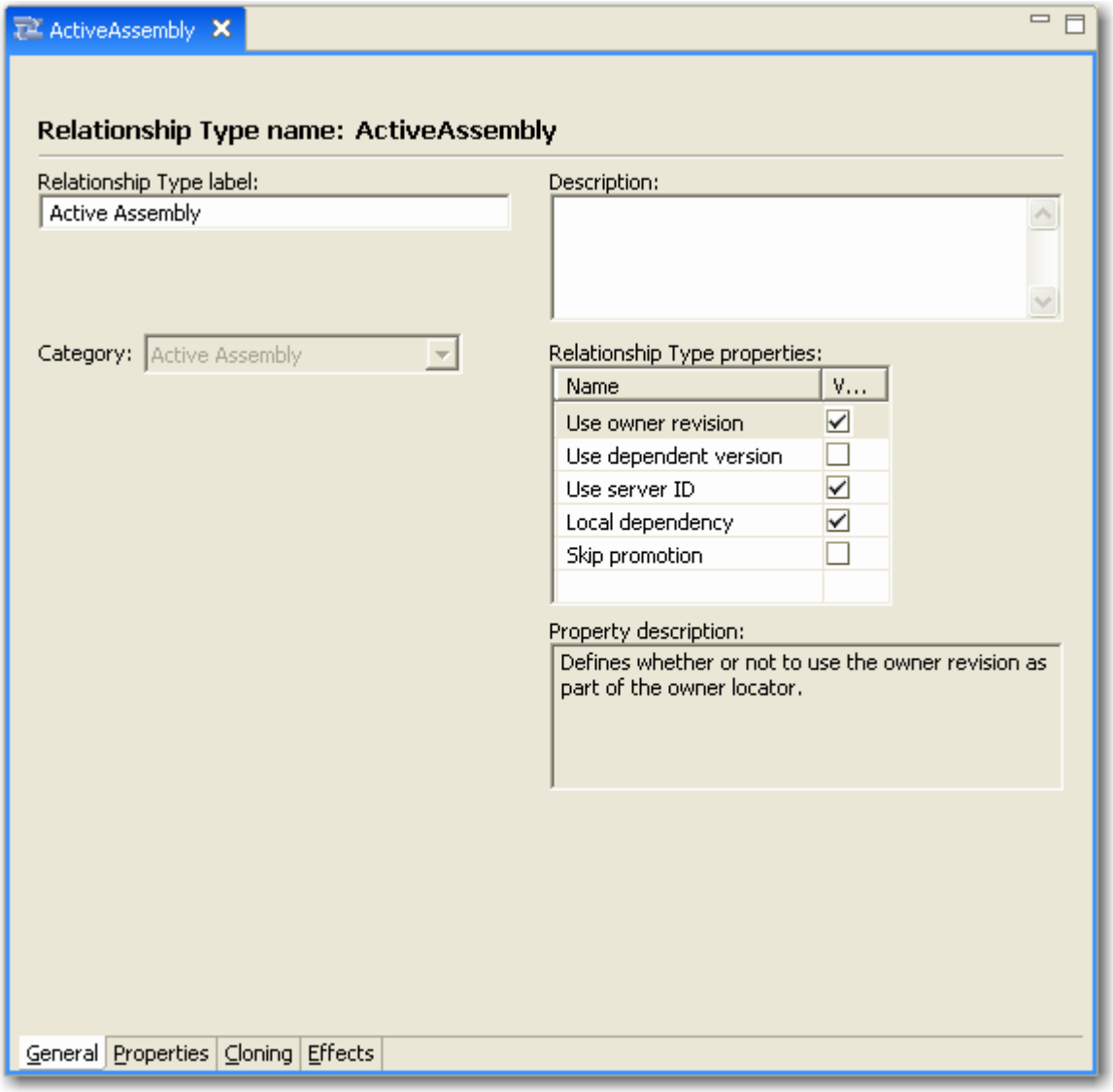

Figure 14: Relationship Type editor, General tab

Field Definitions

Relationship Type label - Required. Name displayed to users for Relationship type.

Category - The category of the Relationship. Read-only for system Relationships. For a list of default categories, see ***New Relationship Type wizard*** (see page 20).

Description - Description of Relationship type.

Relationship Type Properties

Use owner revision - Specifies whether to use the owner's revision ID as part of the owner locater key.

Use dependent revision - Specifies whether to use the dependent's revision ID as part of the dependent locater key.

Use server ID - Specifies whether to use the server ID (rxserver) for executing effects. If set to *No*, the current user is used instead of the server ID.

Local dependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.)

Skip promotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed.

Property Description - Read only. Description of the selected Relationship Type Property.

Relationship Type Editor, Properties Tab

The Relationship Type editor Properties tab displays custom properties assigned to the Relationship Type. Use these properties for processing custom effects.

Note: Effects are extensions that execute during a Relationship's life cycle. For more information about effects, see the document *Implementing the Relationship Engine*.

Use this tab to:

- ***add custom properties to the Relationship Type*** (see "Adding Properties to the Relationship Type" on page 35).
- ***edit custom properties of the Relationship Type*** (see "Editing Properties of a Relationship Type" on page 36).

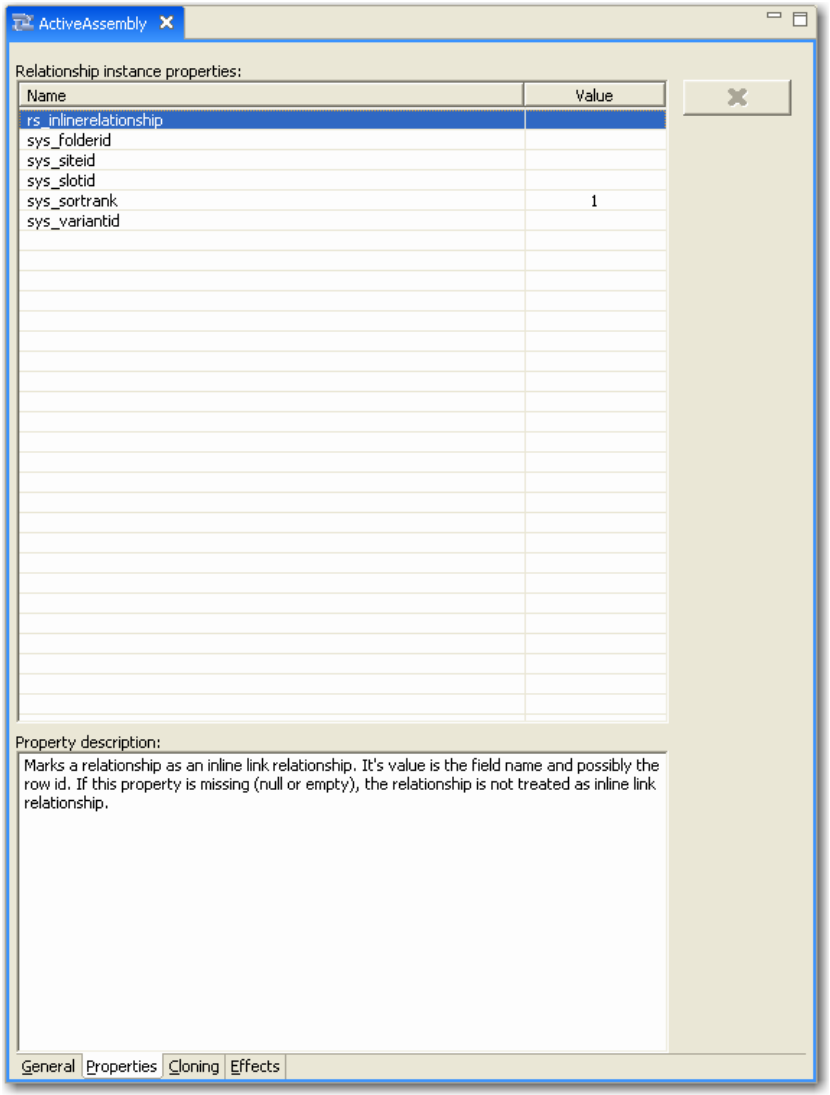

Figure 15: Relationship Type editor Properties tab

Field Definitions

Relationship instance properties - Table of custom properties for Relationships.

Name - The name of the property.

Value - Value of the property, if a value is assigned.

Property Description - Description of the selected Property.

Relationship Type Editor, Cloning Tab

The Relationship Type editor Cloning tab displays rules and conditions for cloning (copying) in the edited Relationship Type.

Use this tab to

- ***set cloning rules and conditions*** (see "Adding Properties to the Relationship Type" on page 35)
- ***edit cloning rules and conditions*** (see "Editing Properties of a Relationship Type" on page 36)

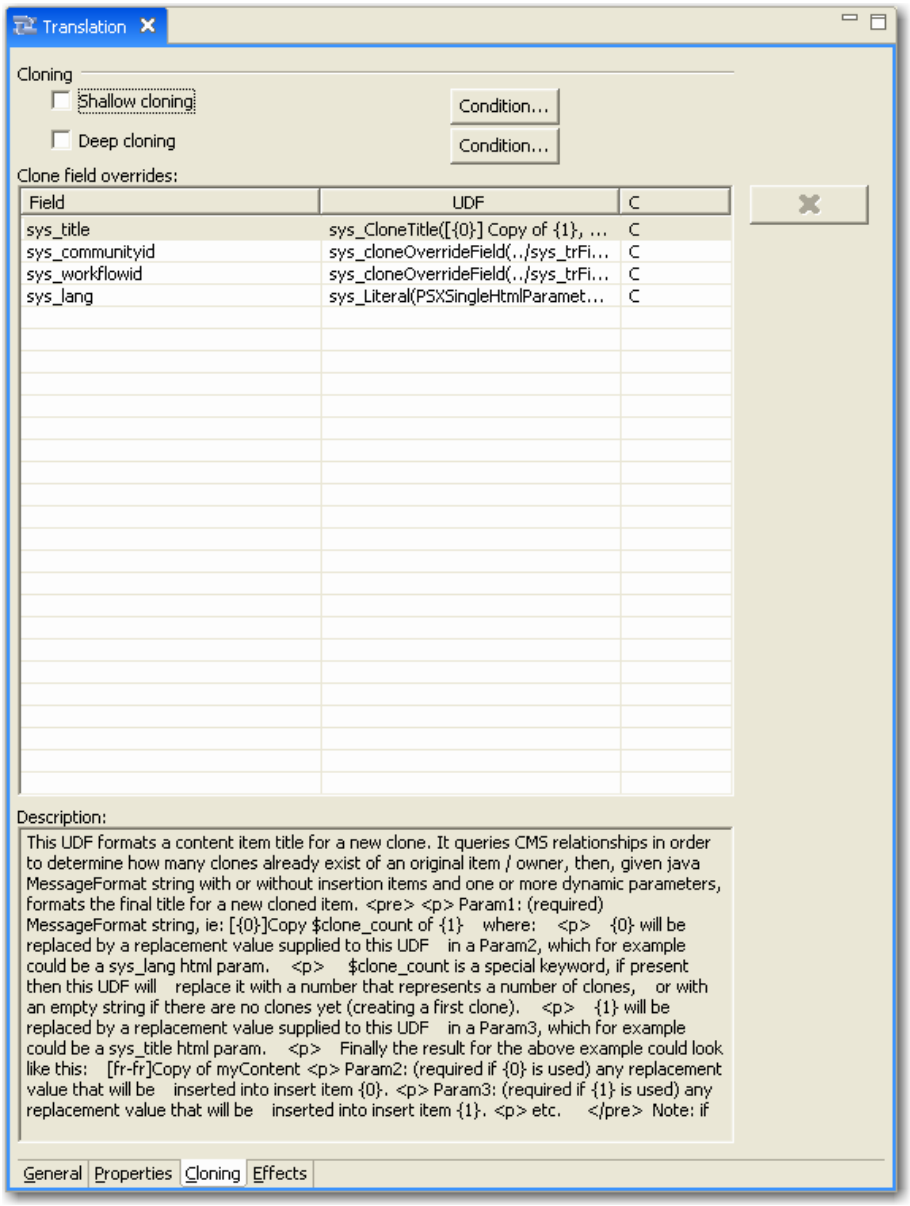

Figure 16: Relationship Type editor Cloning tab

Field Definitions

Shallow cloning - Clone Relationships to dependents, and only make a copy of the original Content Item. In the cloned Relationships, point to the original dependent Content Items.

[**Condition**] - Activates the Rule Editor to specify conditional properties for cloning condition.

Deep cloning - Clone Relationships to descendants (dependents and their dependents, and so on). Make a copy of the original Content Item and each descendant. In the cloned Relationships, point to the copies of the descendants.

[**Condition**] - Activates the Rule Editor to specify conditional properties for cloning condition.

Clone field overrides - The values in this table define which system fields have new values set in the cloned Relationship and how that value is set.

Field - Specifies a field for which to set a new value. Clicking in the field activates a drop list. Options are all system fields.

Udf - Specifies the UDF that is used to define the new value for the field. Options are all UDFs registered in the system.

C - Activates the Rule Editor to specify conditional properties for activating the field override.

Description - Read only. Description of the selected clone process.

Relationship Type Editor, Effects Tab

An effect is a Java extension that runs at some point in a Relationship's lifetime (for example, when a certain transition occurs, an effect may cause users in certain Roles to be notified). The Relationship Type editor Effects tab lists the effects associated with a Relationship and the conditions (contexts) under which they are executed.

Use this tab to:

- ***Add the effects associated with a Relationship and the conditions under which they are executed.*** (see "Adding Properties to the Relationship Type" on page 35)
- ***Edit the effects associated with a Relationship and the conditions under which they are executed.*** (see "Editing Properties of a Relationship Type" on page 36)

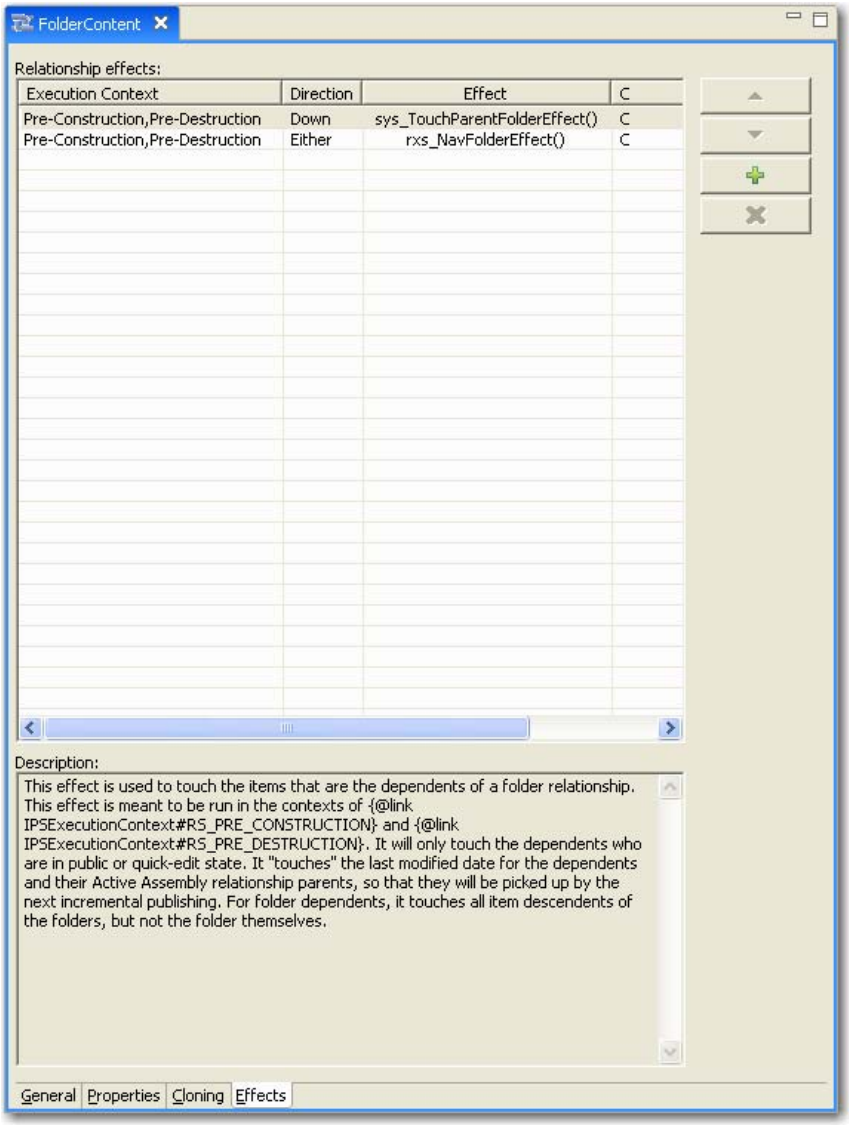

Figure 17: Relationship Type editor Effect tab

Field Definitions

Relationship effects - Table of effects for this Relationship.

Execution Context - Condition that triggers execution of the effect. Click [...] to choose the execution context in the ***Relationship Effects Execution Context dialog*** (see "Relationship Effects Execution Contexts Dialog" on page 30).

Direction - Specifies when to trigger the Effect. Options are: *Down* (triggers the Effect only when the Content Item activating the Effect is the Owner in the Relationship), *Up* (triggers the Effect only when the Content Item activating the Effect is the Dependent in the Relationship), and *Either* (triggers the Effect regardless of which Content Item activated the Effect).

Effect - The name of the Effect. When you click in a blank field, Rhythmyx displays a drop list showing all Effects not currently associated with the Relationship. When you click [...], the Exit Properties dialog opens.

See the document *Implementing the Relationship Engine* for information about the default effects that Rhythmyx provides.

C Click on the  icon to activate the Rule Editor to specify conditional properties for running the effect.

Description - Read only. Displays the description of the selected Effect.

Relationship Effects Execution Contexts Dialog

The Relationship Effects Execution Contexts dialog allows you to specify the Execution Contexts in which an Effect will be invoked for the Relationship. Execution Contexts determine when the associated Effect will be invoked. The following Execution Contexts are available:

- Pre-Construction

  The Effect will be invoked before a Relationship of this Relationship Type is constructed.
- Pre-Destruction

  The Effect will be invoked before a Relationship of this Relationship Type is destroyed.
- Pre-Update

  The Effect will be invoked before the data for a Relationship of this Relationship Type is updated.
- Pre-Clone

  The Effect will be invoked before creating a clone of a Content Type that includes a Relationship of this Relationship Type.
- Pre-Workflow

  The Effect will be invoked before executing a Workflow Transition
- Post-Workflow

  The Effect will be invoked after executing a Workflow Transition.
- Pre-Checkin

  The Effect will be invoked before checking in a Content Item that includes a Relationship of this Relationship Type.

- Post-Checkout

  The Effect will be invoked after checking out a Content Item that includes a Relationship of this Relationship Type.

To access this dialog. click in the Execution Contexts column in a row on the ***Effects*** (see "Relationship Effects Execution Contexts Dialog" on page 30) tab of the Relationship Type editor, then click the browse button.

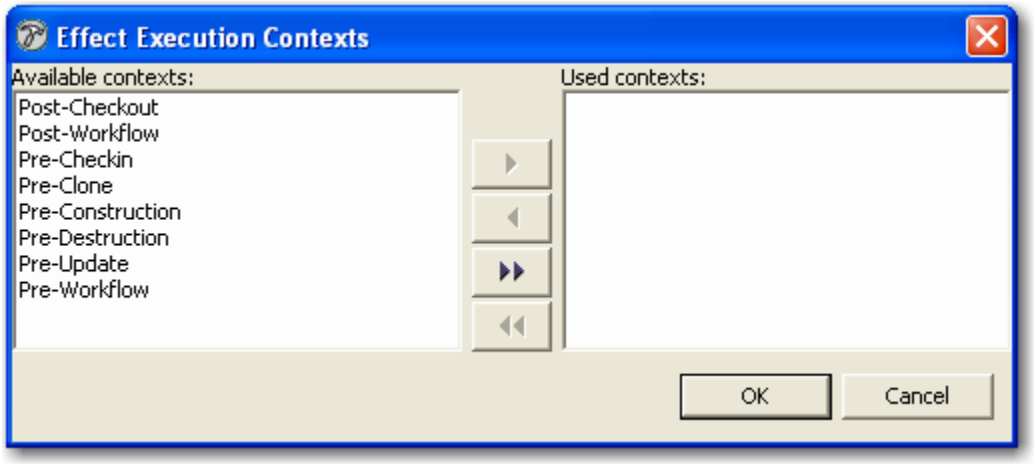

Figure 18: Relationship Effects Execution Contexts dialog

Field Definitions

Available contexts table - lists all unused Contexts.

Used contexts table - lists all Contexts in which the Relationship will be used.

Use the arrow keys to move Contexts to or from the Used Contexts table. Click [**OK**] to save the Used Contexts and return to the ***Effects*** (see "Relationship Effects Execution Contexts Dialog" on page 30) tab of the Relationship Type editor

Rule Editor

Use the Rule Editor to specify the conditions that trigger an Exit or Effect, or that permit a specific type of Cloning.

You can write simple Rules in the Rule Editor itself. For example, you can write a rule that tests an HTML parameter against a literal value right in the Rule Editor. However, more complicated Rule, particularly Rules that require reference to other objects in the system, may require an Extension. For example, if you wanted to evaluate whether a Slot contains a certain number of Content Items, the Rule Editor does not have the facilities to perform the check. You would have to write an extension to evaluate this rule. The extension must be a UDF that generates a boolean value (in other words, either TRUE or FALSE.

To access the Rule Editor, double click on the  icon in the C column on the Cloning, Exits, or Effects panel of the Relationship Editor.

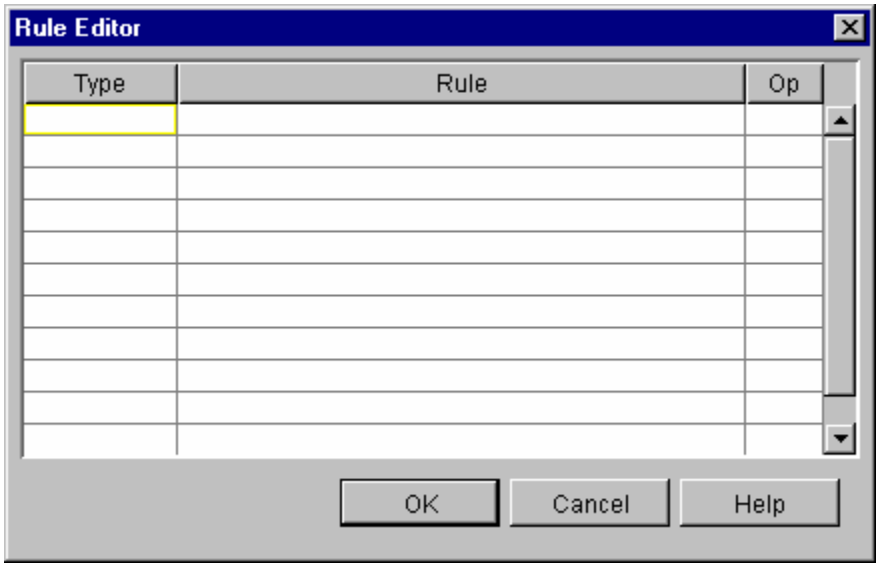

Figure 19: Rule Editor

Columns

Type Drop List. Specifies the type of Rule. Two options are available: *Condition* and *Extension*.

Rule Defines the Rule. If the **Type** is *Condition*, you must specify the conditions to me met (for example, psx-locale=fr=fr) in this column. Use the Conditional Property dialog to specify the conditions. If the **Type** is *Extension*, you must specify the extension to produce the result. The extension must result in a boolean value (in other words, either TRUE or FALSE).

Op Specifies a boolean operator to join multiple rules. Options are *AND* or *OR*, or null.

CHAPTER 3

Maintaining Relationship Types

You can ***create*** (see page 34), ***modify*** (see page 36), or ***delete*** (see page 37) any Relationship under the User node of the Relationship Editor. You can modify Relationships under the System node, but you cannot add new Relationships to this node or delete Relationships from it.

Most Relationships define an association between Content Items. If you want users to be able to act on the Relationship, you need to create a new Action Menu Entry for the Relationship. For details, see "Customizing Action Menus" in *Implementing the Rhythmyx Business User's Interface*.

Creating a Basic Relationship Type

To enter the minimum amount of information to create a Relationship Type that appears in System Design view, complete the New Relationship Type wizard.

For a graphic of the wizard and definitions of the fields discussed below, see ***New Relationship Type Wizard*** (see page 20).

To create a Relationship Type:

1. In System Design view, right-click on the Relationship Types folder and choose *New* > *Relationship Type*.

   The New Relationship Type wizard opens.

2. In Relationship Type name, enter an internal name for the Relationship Type.

   The Relationship Type name is automatically entered in Label.

3. Optionally, change the display name in Label.

4. Optionally, in Description, enter a description for the Relationship Type.

5. In Category, choose a category for the Relationship.

6. Click [**Finish**].

   The Relationship Type is created and appears in the Relationship Types section of the System view. The New Relationship Type wizard closes and the ***Relationship Type editor*** (see page 22) opens.

Now you can:

- ***configure properties of the Relationship Type*** (see "Adding Properties to the Relationship Type" on page 35) in the Relationship Type editor.
- close the Relationship Type editor, and use the default properties for the Relationship or continue configuring the Relationship Type at a later time.

Adding Properties to the Relationship Type

For graphics of the Relationship Type editor, instructions on opening it, and definitions of the fields discussed below, see the ***Relationship Type Editor*** (see page 22) section including the topics on each tab.

To add properties to a Relationship:

1. In the Relationship Type editor, click the ***General tab*** (see "Relationship Type Editor, General Tab" on page 23).

   Check any of the Relationship Type properties that you want to apply to the Relationship Type. Uncheck any of the Relationship Type properties that are checked by default.

2. Click the ***Properties tab*** (see "Relationship Type Editor, Properties Tab" on page 25). Add any custom properties to be used when processing effects. For each property enter a Name, and Value, and optionally a Property Description.

3. Click the Cloning tab. To clone the Relationship when an item is copied, check Allow cloning.

   a) To allow shallow cloning, check Shallow cloning. To include a condition when Shallow cloning occurs, click [**Condition**] and fill in the Rule editor.

   b) To allow deep cloning, check Deep cloning. To include a condition when Deep cloning occurs, click [**Condition**] and fill in the Rule editor.

   c) To set new values for system fields in the cloned copies, fill in the Clone field overrides table:

      - In the Field column, choose a system field.
      - In the Udf column choose a UDF to define the new value for the field.
      - Click in the C column to open the Rule Editor for specifying the condition under which the field is overridden with the value that the Udf generates.

4. Click the Effects tab. Enter effects for the Relationship Type in the Relationship effects table. For each effect that you enter:

   d) In Execution Context, click [**...**] to display the ***Relationship Effects Execution Context dialog*** (see "Relationship Effects Execution Contexts Dialog" on page 30) and move the Execution Contexts in which you want to invoke the Effect for Relationships of this Relationship Type from the Available contexts field to the Used contexts column.

   e) In Direction, choose when to trigger the effect.

   f) In Effect, choose an effect. Click [...] to open the Exit Properties dialog. Add parameters for the effect if required.

   g) In C, click on the  icon to activate the Rule Editor to specify conditional properties for running the effect.

5. Save and close the Relationship Type editor.

Now you can:

- Associate the Relationship Type with a Menu Entry.

Editing Properties of a Relationship Type

When you edit a Relationship Type, you can modify any of the information accessible from the Relationship Type editor.

For graphics of the Relationship Type editor and definitions of the fields discussed below, see the ***Relationship Type Editor*** (see page 22) section including the topics on each tab.

To edit a Relationship Type:

1. Right-click on the Relationship Type object in the System Design View and select *Open.*

   The Relationship Type editor opens.

2. On the ***General tab*** (see "Relationship Type Editor, General Tab" on page 23), you can change:

   Relationship Type label

   Category (you cannot change the Category of a System Relationship)

   Description

   Relationship Type properties

3. On the ***Properties tab*** (see "Relationship Type Editor, Properties Tab" on page 25) you can:
   - Change the Name, Value, Type, and Property Description of any property.
   - Choose the row for any property and click [**X**] to delete it.

4. On the Cloning tab you can:
   - Check or uncheck Shallow cloning and/or click [**Condition**] to edit the condition under which it is applicable in the Rule Editor.
   - Check or uncheck Deep cloning and/or click [**Condition**] to edit the condition under which it is applicable in the Rule Editor.
   - Add, edit, or remove Clone field overrides for system fields. Change any of the values under Field and UDF in the table or click the C column to edit the condition under which the override is applicable in the Rule Editor.

5. On the ***Effects tab*** (see "Relationship Type Editor, Effects Tab" on page 29) you can:
   - Change the Execution Context, Direction or Effect for any effect or click the C column to edit the condition under which the effect executes in the Rule Editor.
   - Add an effect in the first empty row. See ***Adding Properties to a Relationship Type*** (see "Adding Properties to the Relationship Type" on page 35) for information on adding an effect.
   - Select an effect and choose [^] or [**V**] to change the order in which it is executed.
   - Select an effect and choose [**X**] to remove it.

6. Save and close the editor.

Deleting a Relationship Type

To delete a Relationship Type:

1. In System Design view, right-click on the Relationship Type and choose *Delete*.

   The Relationship Type is deleted.

Defining Conditions for Exits, Effects, and Cloning Processes

When adding an an Effect to a Relationship, you can specify conditions that trigger that extension. You can also define conditions for both the deep and shallow cloning processes.

To define conditions for an extension or cloning process:

1. Double-click the  icon in the row of the Effect or cloning process to which you want to add conditions.

   Rhythmyx displays the Rule Editor.

2. To add a Rule as a Condition:

   a) In the first blank row on the Rule Editor, click in the Type field and choose *Conditional*.

   b) Double click in the Rule column of the same row to activate the Rule field.

   c) Click on the browse button (...).

   d) Rhythmyx displays the Conditional Properties dialog.

   e) Click in the Variable column, then click the browse button to display the Value Selector. Specify the Value for the Variable.

   f) In the Op column, choose an operator.

   g) Click the Value column, then click the browse button to display the Value Selector. Specify the Value for the Value.

   h) If you want to add another condition, click in the bool field and choose the boolean operator for the additional condition. Options are *AND* and *OR*. Note that if you add multiple conditions on this dialog, they are treated as a single Rule on the Rule Editor. In other words, the result of the entire set of conditions is treated as the result of the Rule.

   i) Click [**OK**] to save the condition.

3. To specify an Exit to process the Rule:

   a) In the first blank role on the Rule Editor, click in the Type field and choose *Extension*.

   b) Double-click in the Rule field of the same column and select the extension you want to use for the Rule. The extension should be a UDF that generates a boolean result (in other words, eithe TRUE or FALSE).

4. If you want to add another Rule, click in the Op column of the Rule and choose the boolean operator you want to use to process the additional rule. Options are *AND* and *OR*.

5. Click [**OK**] to save your rules.

Planning Clone Field Overrides

Before implementing clone field overrides, decide which fields you want to override and how you want them to change. Only system fields (fields defined in the ContentEditorSystemDef.xml) are eligible for override; fields defined in shared and local definition XML files are not eligible to be overridden.

Some typical overrides are:

Name	Internal Name	Common Change
Community	sys_communityid	Change the ID to put the Content Item in a new Community.
Workflow	sys_workflowid	Change the ID to put the Content Item into a different Workflow. (You generally don't have to change the State because clones are always created in the initial State of any Workflow.)
Locale	sys_lang	Change the Locale of the Content Item.
Title	sys_title	Add some form of increment to the title to indicate where in the sequence of clones it falls. For example, you may want the title to include the phrase "Copy X of Y copies".

Rhythmyx ships with two UDFs that can perform these simple overrides:

- sys_cloneFieldOverride

  This UDF calls a Rhythmyx resource that generates an XML document from which you can derive new values for a field.

- sys_CloneTitle

  This UDF adds text to the title of the clone indicating where in the sequence of clones it falls; for example, "Copy X of Y".

If you want a more complicated override than these UDFs provide, you will need to write your own UDF to perform the override processing.

CHAPTER 4

Modifying Relationship Configurations

The Rhythmyx installation provides a number of default Relationships that help the system operate. These default Relationships should generally meet your needs, but you will probably want to reconfigure them to match the functionality you want in your Content Management System.

Simple Reconfiguration: Adding Forced Transition to a Mandatory Relationship

To illustrate a simple reconfiguration, let us modify the default Active Assembly – Mandatory Relationship to force associated Content Items to Public.

Mandatory Relationships are Relationships in which both Content Items in the Relationship must go Public together. Mandatory Relationships are implemented through the sys_PublishMandatory and sys_UnpublishMandatory Effects. These Effects provide the processing that determines whether the Content Items in a Relationship can go Public.

In the default Active Assembly – Mandatory, these are the only two Effects assigned. The Direction configured for each is Down. In other words, if a Content Item has an Active Assembly – Mandatory Relationship to another Content Item, when the Owner in the Relationship makes a Transition to Public, this Effect will be triggered to check whether the Dependent in the Relationship is Public. If the Dependent makes this Transition, the Effect does not check on the current State of the Owner.

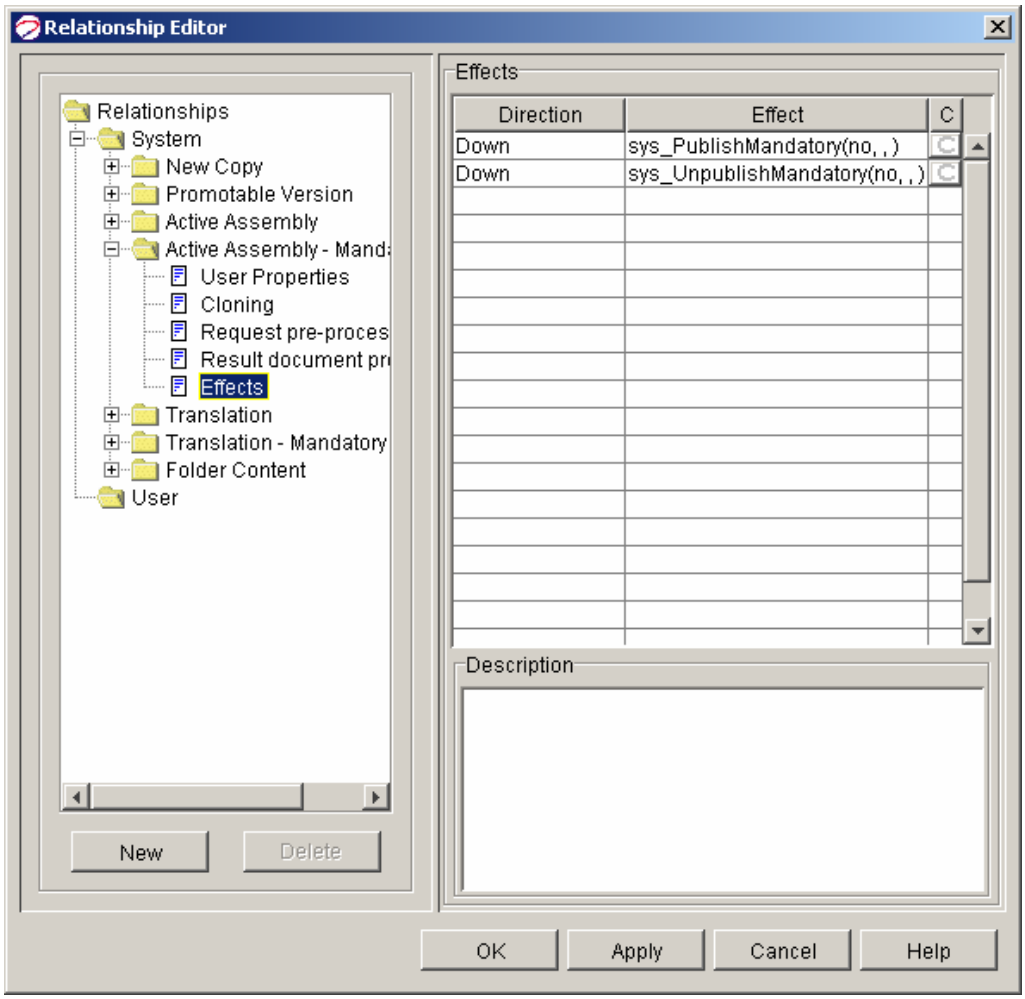

Figure 20: Effects Configured for the Active Assembly - Mandatory Relationship

Let us focus on the sys_PublishMandatory Effect. This Effect has three parameters:

Parameter	Description
forceTransition	Mandatory. Specifies whether to try to force the other Content Item in the Relationship to make a Transition if it is not currently Public.
ownerTransitionName	Optional. Specifies the name of the Transition to use to force the Owner in the Relationship if it is not already Public.
dependentTransitionName	Optional. Specifies the name of the Transition to use to force the Dependent in the Relationship if it is not already Public.

In the default configuration, the value of the forceTransition parameter is *no*, meaning that the dependent is not forced to Public when the Owner is Transitioned to Public. In this configuration, the other two parameters are irrelevant, so they are blank.

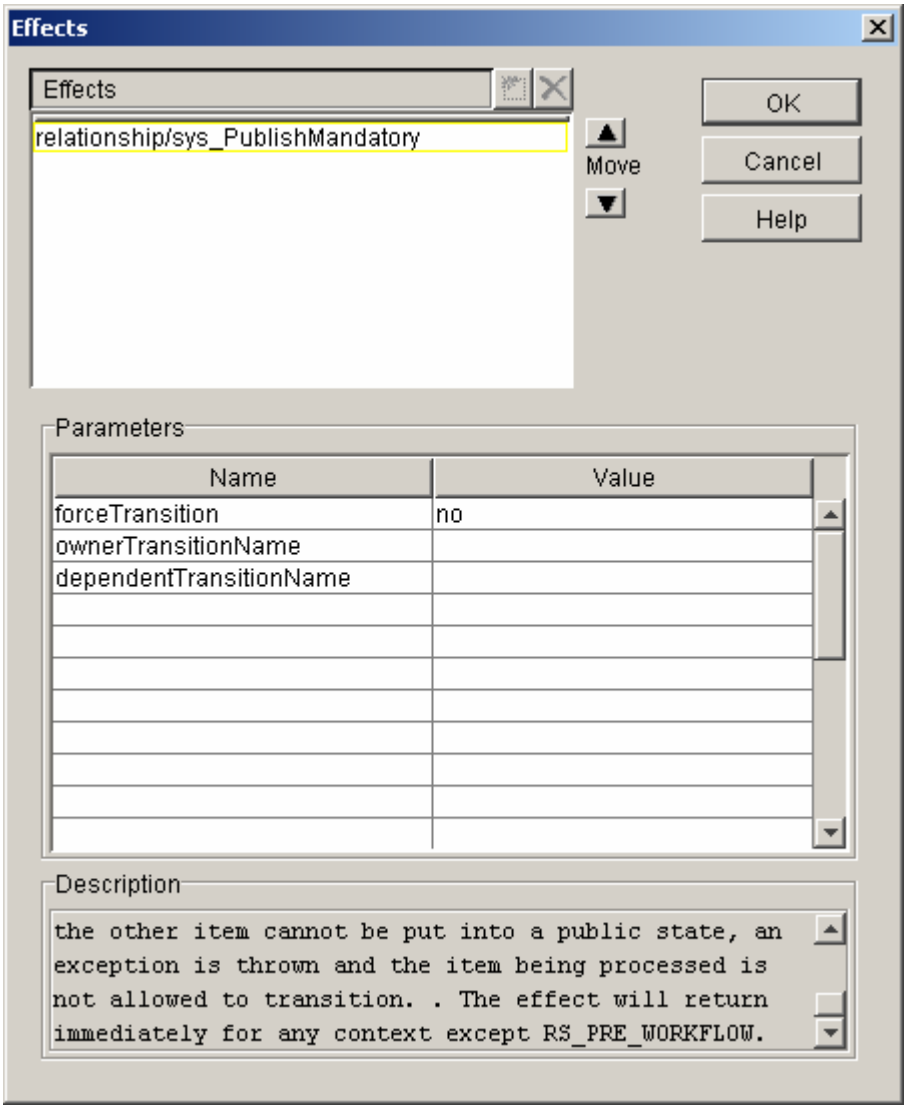

Figure 21: Default Configuration of the sys_PublishMandatory Effect of the Active Assemby - Mandatory Relationship

Let us now suppose that whenever a Content Item has an associated graphic, we want to force the graphic to go Public with its owner. The Active Assembly – Mandatory Relationship provides this functionality, but we will need to modify the configuration, changing the value of the forceTransition parameter to yes, and entering the name of the Transition we want to use to force the Dependent Public.

Since we want to use this Relationship for graphics, let us examine the Images Workflow:

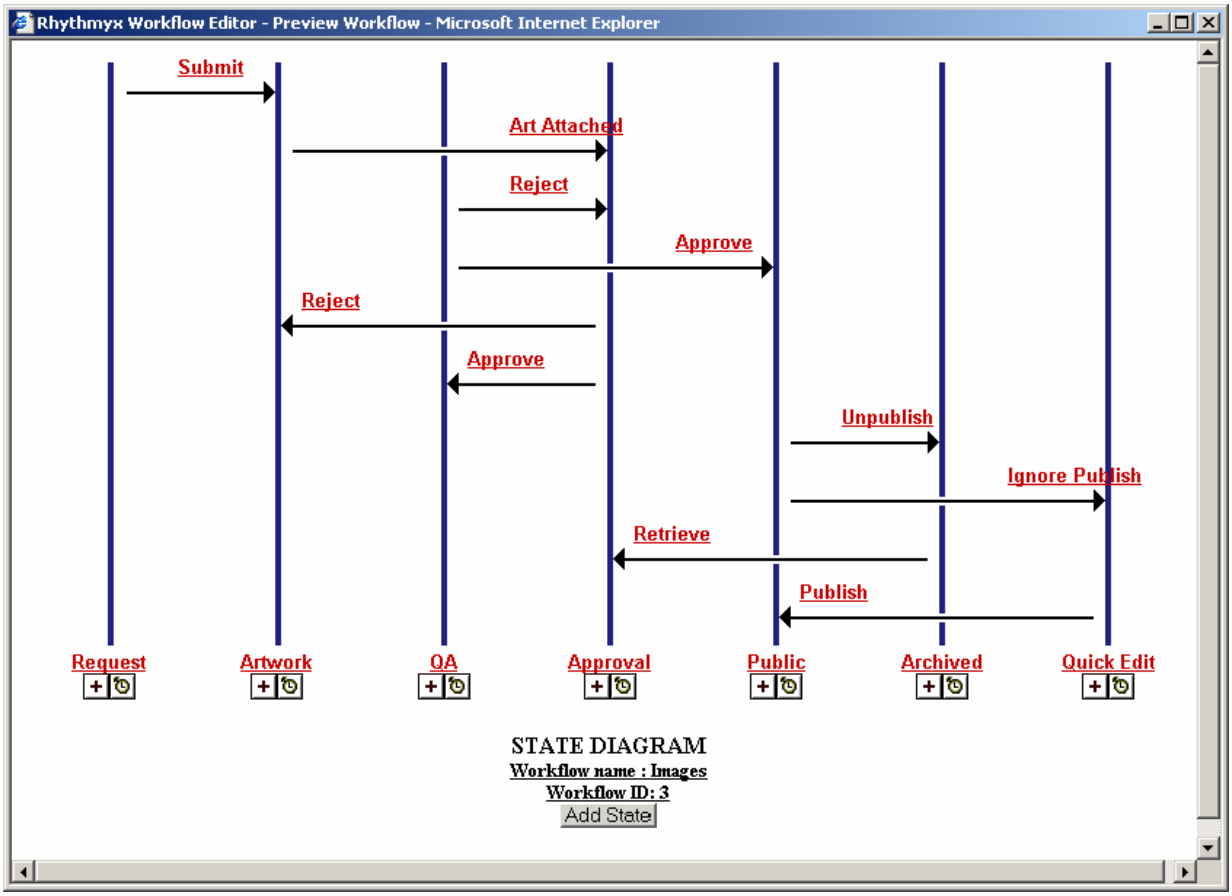

Figure 22: Images Workflow

In this Workflow, two Transitions move Content Items to the Public State:

- The Approve Transition from the QA State
- The Publish Transition from the Quick Edit State

We can ignore the latter Transition, because Quick Edit is a special State used for minor edits to Content Items that are already Public. This fact leaves us with the Approve Transition to the Public State. We will enter Approve as the value of the dependentTransitionName parameter of the sys_PublishMandatory Effect.

(Note that sys_PublishMandatory Effect generates an error if a Transition cannot move a Content Item to Public. Thus, we do not need to be concerned that there is another Transition with the name “Approve”. This Transition moves a Content Item from the Approval State to the QA State, and would cause an error if Rhythmyx attempted to force this particular Transition. Only the Approve Transition from QA to Public will move a Content Item.)

The following graphic illustrates the reconfigured Effect:

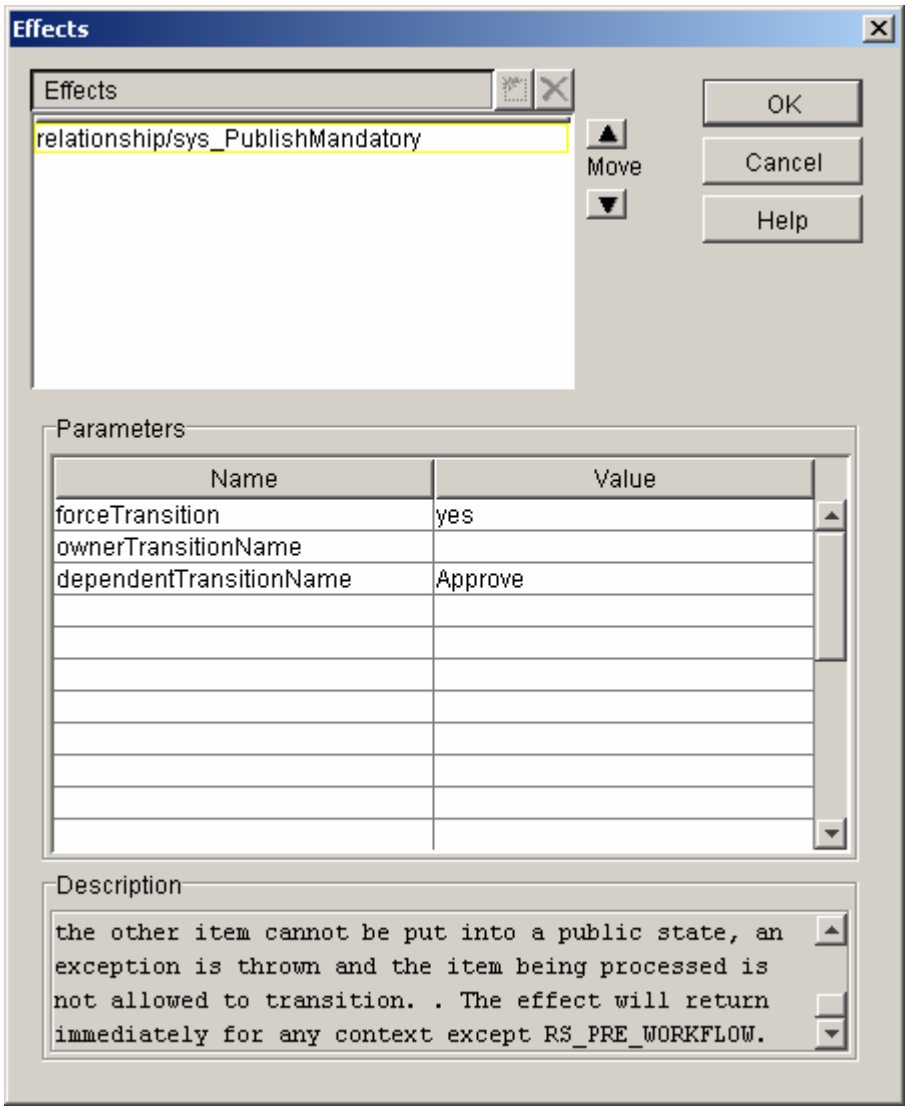

Figure 23: Reconfiguration of the sys_PublishMandatory Effect to force a Dependent Content Item to Public using the Approve Transition

The sys_UnpublishMandatory Effect would be reconfigured in the same fashion.

Advanced Reconfiguration: Conditional Cloning Based on the Locale of a Translation

More advanced reconfiguration may require conditional processing to determine whether cloning occurs or an Exit or Effect is triggered. Let us examine the reconfiguration required to facilitate the processing described in the ***Advanced example of Relationship processing*** (see "Advanced Example: Translations" on page 10). In this example, when we create a Translation, we want to change the conditions under which we clone Active Assembly Relationships. Thus, even though we are reconfiguring to facilitate Translations, we will be modifying the configuration of the Active Assembly Relationship. Specifically, we will be modifying the Cloning Properties.

Let us begin with the default configuration of the Active Assembly Relationship.

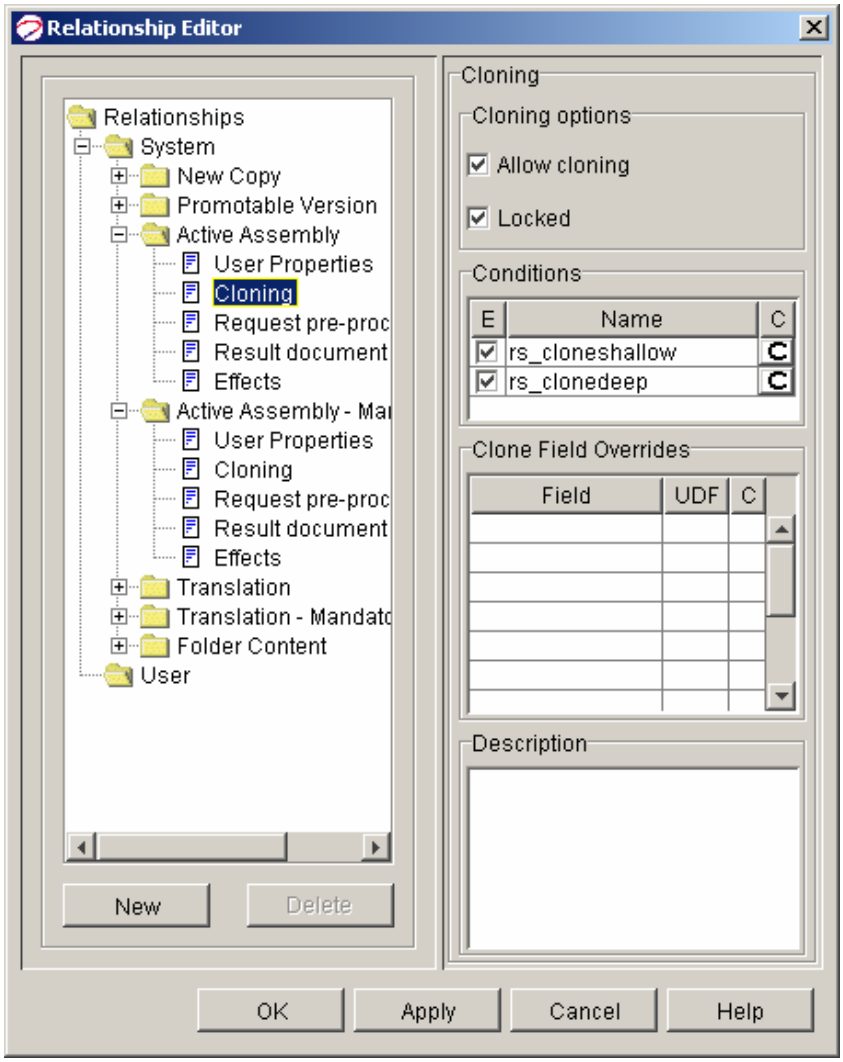

Figure 24: Default Cloning Configuration of the Active Assembly Relationship

Cloning is enabled for this Relationship, and both shallow cloning (cloning only the Relationship itself) and deep cloning (cloning the Relationship and the Dependent Content Item in the Relationship) are allowed. Conditional processing has been defined for both shallow and deep cloning. For shallow cloning, a single condition has been defined, processed in the Rule Editor:

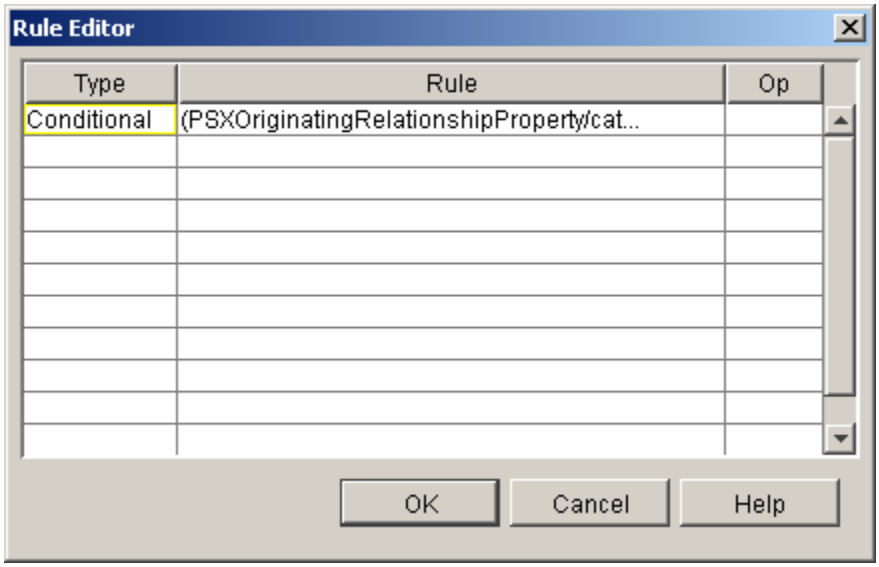

Figure 25: Default ShallowCloning Conditions for the Active Assembly Relationship

The following specific rule has been defined:

Variable	Operator	Value	Boolean
PSXOriginatingRelationshipProperty/Category	=	rs_promotable	OR
PSXOriginatingRelationshipProperty/Category	=	rs_copy	

If category of the Originating Relationship (the Relationship that is triggering the creation of the clone) is either Promotable (category=rs_promote) or Copy (category=rs_copy), then Rhythmyx will make a shallow clone (clone only the Active Assembly Relationship).

For deep cloning, we also have a single condition. The rule for this condition is:

Variable	Operator	Value	Boolean
PSXOriginatingRelationshipProperty/Category	=	rs_translation	

If the Category of the Originating Relationship is Translation (category=rs_translation), then Rhythmyx will create a deep clone (clone both the Active Assembly Relationship and the Dependent Content Item in the Relationship).

In the example, we stated that we will use the same graphic in the Canadian French Locale that we use in the default US English Locale. Therefore, when Translating to the Canadian French Locale, we can use shallow cloning: when we create a Translation for the Canadian French Locale, we will create a new Relationship that points to the same Dependent Content Item that is used in the US English Locale.

To accomplish this goal, we can add another condition to the shallow cloning condition.

If the Category of the Originating Relationships is Translation AND

If the Locale of the new Content Item is Canadian French.

You must use the internal value for the Relationship Category. The internal value for the Translation Category is rs_translation. The locale is stored in the HTML parameter sys_lang. (Note that while the variable sys_lang is also available as Content Item Data, we are not actually working with a Content Item when cloning a Relationship, so we must use the HTML parameter.)

Translated into Rhythmyx terms, we will have the following Rule:

Variable	Operator	Value	Boolean
PSXOriginatingRelationshipProperty/Category	=	rs_translation	AND
PSXParam/sys_lang	=	fr-ca	

We could add these additional Rules to the existing Rules, but since it is a more complex statement, it makes more sense to add it as a new Rule. Since we now have multiple Rules, we need to define some Boolean processing for them. We cannot use the AND connector between these two Rules. If we used that connector, the Originating Relationship would have to be in two Categories (Copy and Translation; or Promote and Translation), and a Rhythmyx Relationship can have only one Category. Thus, we must use the OR connector. The following graphic illustrates the final configuration:

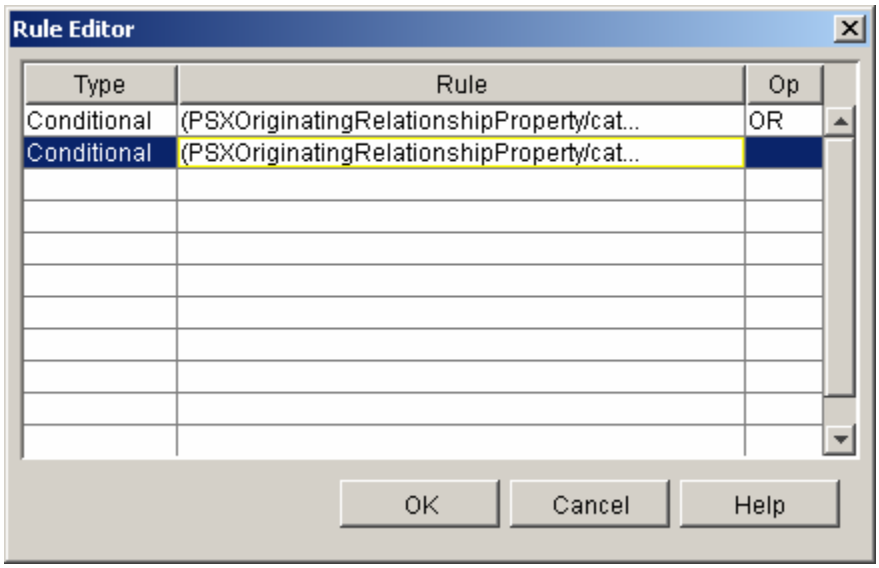

Figure 26: New ShallowCloning Conditions for the Active Assembly Relationship

Next, we need to address the Japanese Locale. When translating to the Japanese Locale, we want a new graphic. Thus, we need a deep clone, which copies both the Active Assembly Relationship and the Dependent Content Item in the Relationship.

As we noted above, the default conditions for deep cloning the Active Assembly Relationship already call for deep cloning when the category of the Originating Relationship is Translation. We want to add another rule to this condition, specifying that, in addition to the existing rule, the Locale of the Translation must be Japanese (ja-jp):

Variable	Operator	Value	Boolean
PSXOriginatingRelationshipProperty/Category	=	rs_translation	AND
PSXParam/sys_lang	=	ja-jp	

This configuration is more restrictive than the default configuration. In the default configuration, any clone created when the originating Relationship was in the Translation Category would be a deep clone. Now, only clones to the specified ja-jp Locale will be deep cloned. If you want to allow deep cloning in other Locales, you will need to add more Rules defining those Locales. If neither condition is evaluated as TRUE (in other words, if we create a Translation Content Item in a locale other than fr-ca or ja-jp), then no clone of the Active Assembly Relationship is created.

CHAPTER 5

Overriding Content Item Fields in Clones

When you clone a Content Item (such as to create a new Translation Content Item), you frequently want to change the value in a field on the clone. When a Business User creates the clone manually, the user can modify the field in the clone. When you create a clone automatically, such as automatically generating a Translation Content Item, you probably want to change the value in certain fields automatically in the process. Use Clone Field Overrides to automate these changes. Clone field overrides can automatically update fields in a clone with new values.

When defining a Relationship type in the Relationship Editor, you can specify the fields you want to override. The override processing itself is performed by a UDF you specify for the field. You can also specify conditional processing to determine whether or not to override the field.

Implementing Clone Field Overrides

To implement a field override:

1. On the Relationship Editor, click the Cloning node.
2. Double-click in the first empty row in the Fields column and select the field you want to override from the drop list. Options are all system fields.
3. Double-click in the UDF column and select the UDF you want to use to modify the field in the clone.

   Rhythmyx displays the UDF Editor.
4. Enter Values for each parameter you want to use in the UDF. Note that some parameters are mandatory while others are optional.
5. Click the [**OK**] button on the UDF Editor.

   Rhythmyx returns you to the Relationship Editor.
6. If you want to add conditions to the field override, click the  button to display the Rule Editor. For details about adding rules, see Defining Conditions for Exits, Effects, and Cloning Processes.
7. Repeat steps 2-6 for all fields you want to override.
8. Click the [**Apply**] button to save your changes.

Example Implementation of Clone Field Overrides

To illustrate how clone field overrides work, let us examine the implementation of the default Translation Relationship shipped with Rhythmyx. The following graphic illustrates the Cloning properties of the Translation Relationship:

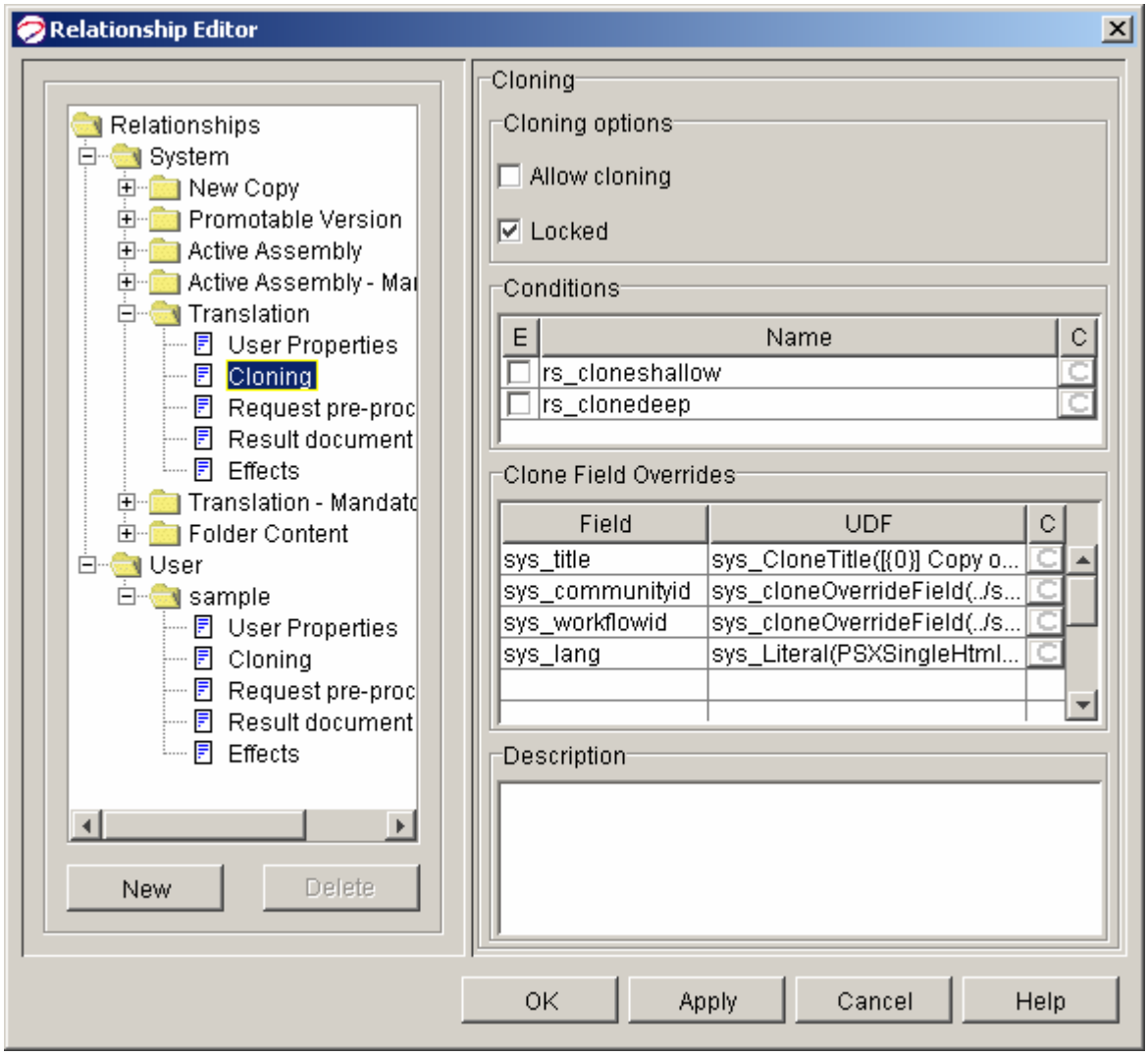

Figure 27: Default Cloning Coniguration of the Translation Relationship

When creating a clone using the Translation Relationship, Rhythmyx will modify the following fields:

- sys_title
- sys_communityid
- sys_workflowid

- sys_lang

Let us examine some of these overrides.

Overriding the sys_title Field

To create a new title, the sys_CloneTitle UDF is applied to the sys_title field.

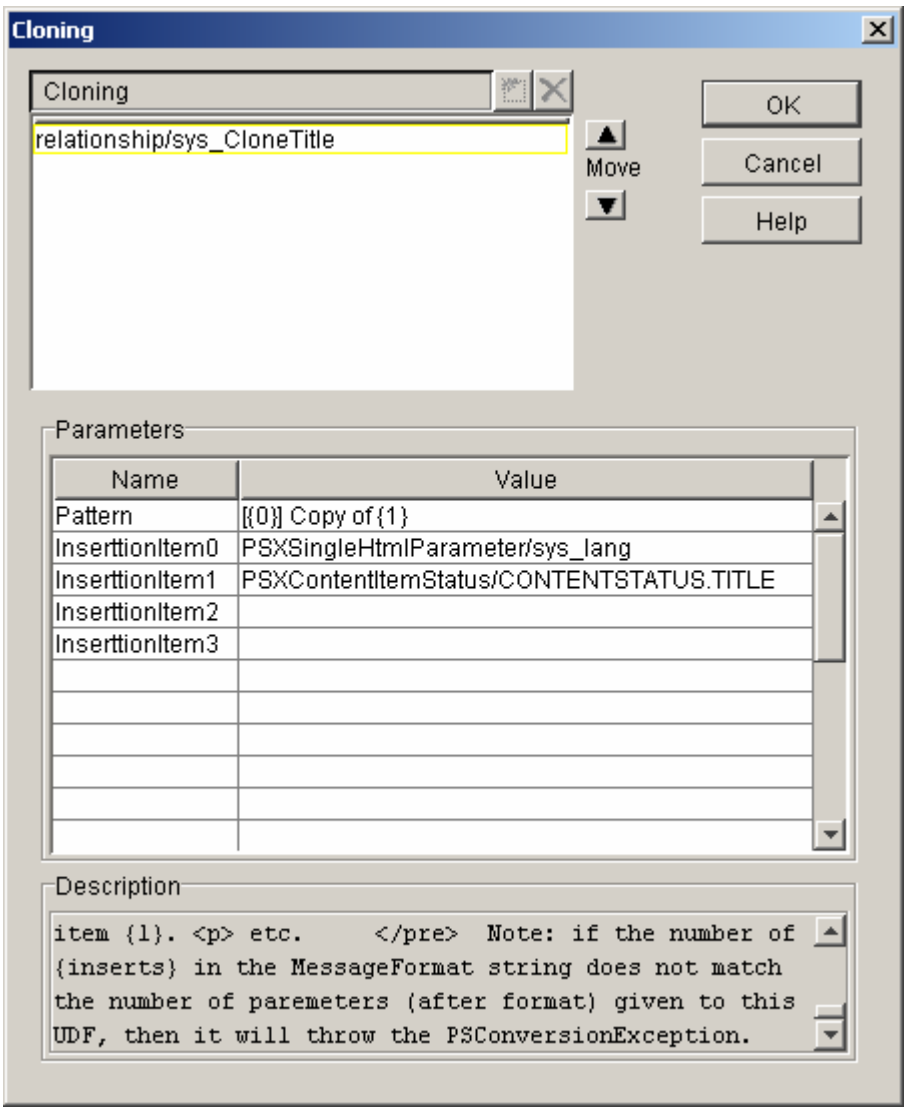

Figure 28: Default Coniguration of the sys_CloneTitle UDF for the Translation Relationship

This UDF creates a new title according to the pattern specified in the Pattern parameter. You can insert any string in this pattern. To add variable values derived from the data in the Content Item, use the variables 0-3 in curly braces (“{}”) to specify the insertion of a value derived from the InsertionItem parameter specified by the numeric value. You can also use the `$clonecount` macro, which will insert the count of this clone among all the clones created from the Owner Content Item.

In this case, the pattern is

```
[{0}] Copy of {1}
```

Since we are working with the Translation Relationship, it might make sense to change this pattern to use the term “translation” rather than “copy”:

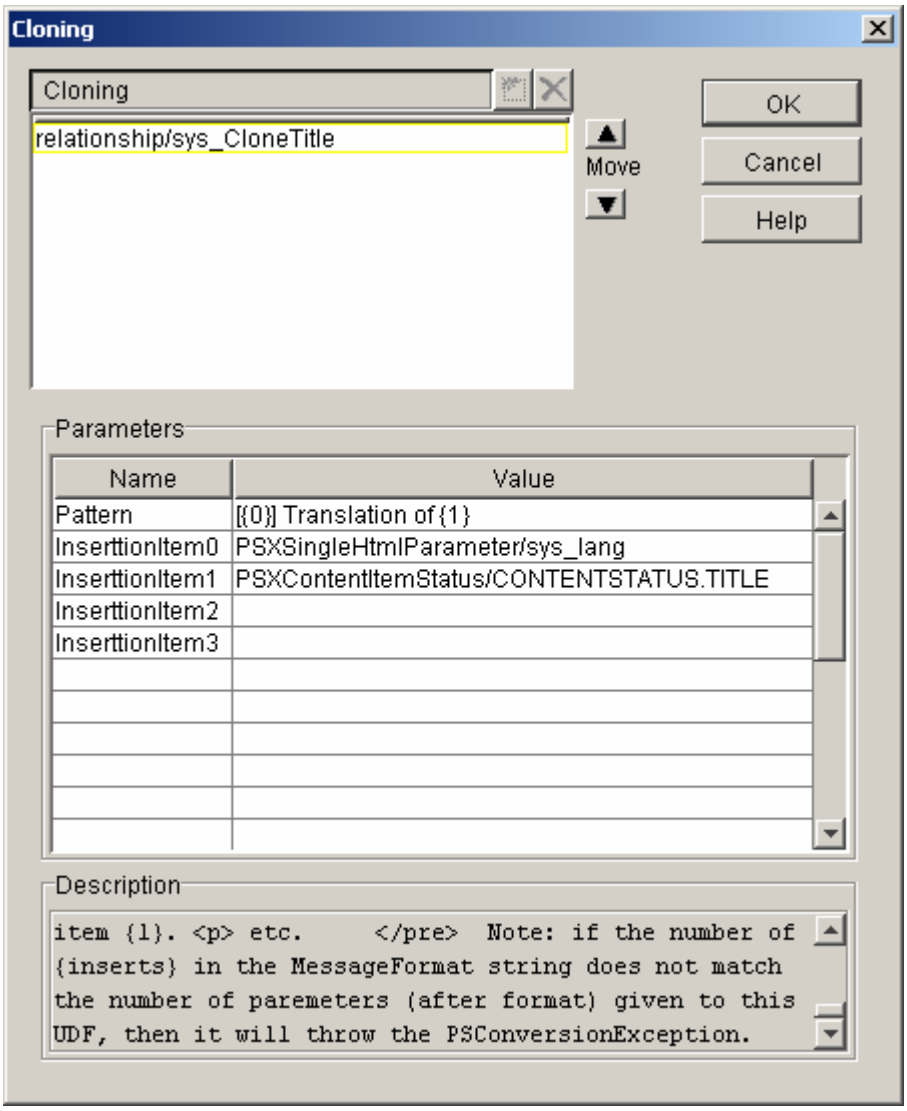

Figure 29: Modifying the Pattern parameter of sys_CloneTitle UDF from "Copy" to "Translation"

The `InsertionItem0` parameter specifies that we will insert the value of the `sys_lang HTML` parameter into the Pattern, and the `InsertionItem1` parameter specifies that we will insert the Title (CONTENTSTATUS.TITLE) of the Owner Content Item.

The final output of this UDF will be something like this: `[es-mx] Translation of Title`.

Overriding the Community Field

To update the Community, we use the sys_cloneOverrideField UDF.

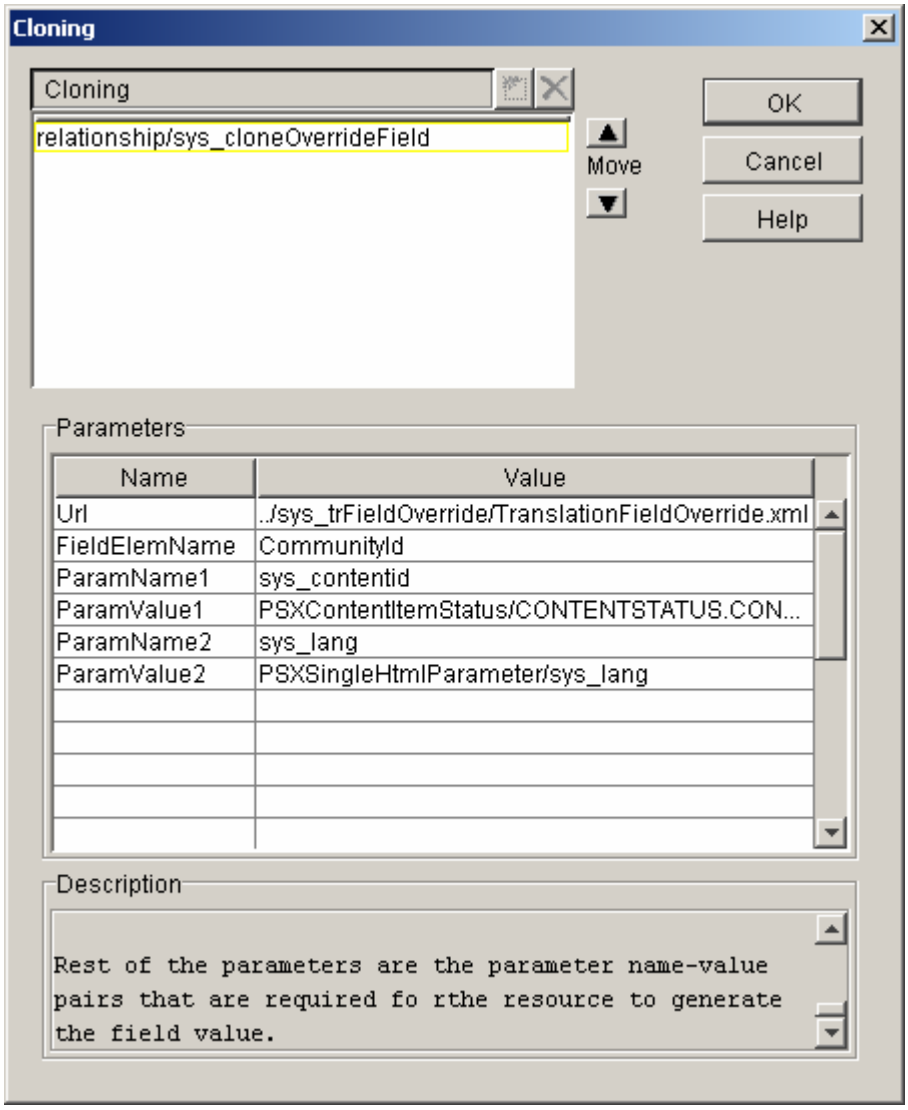

Figure 30: Default Coniguration of the sys_cloneOverrideField DF for theCommunity in the Translation Relationship

This UDF queries a Rhythmyx resource for an XML document, and selects one node of this document as the new value for the field. The Url parameter specifies the resource. In this case the resource we query is ./sys_trFieldOverride/TranslationFieldOverride.xml. This default resource queries the AUTOTRANSLATION table for Translation configuration data. The following table shows the mappings of this resource:

Backend	**XML**
PSX_AUTOTRANSLATION.COMMUNITYID	TranslationFieldOverride/Entry/CommunityId
PSX_AUTOTRANSLATION.CONTENTTYPEID	TranslationFieldOverride/Entry/ContentTypeId

Backend	XML
PSX_AUTOTRANSLATION.LOCALE	TranslationFieldOverride/Entry/Locale
PSX_AUTOTRANSLATION.WORKFLOWID	TranslationFieldOverride/Entry/WorkflowId

The sys_contentid (PSXContentItemStatus/CONTENTSTATUS.CONTENTID) and sys_lang (PSXSingleHtmlParameter/sys_lang) parameters are used in the WHERE table on the Selector for the query:

Variable	Op	Value	Bool
CONTENTSTATUS.CONTENTID	=	PSXSingleHTMLParameter/contentid	AND
PSXAUTOTRANSLATION.LOCALE	=	PSXSingleHTMLParameter/sys_lang	

This UDF definition will extract the value of the CommunityId element from the resulting XML document and insert that value as the Community ID for the new Content Item.

Overrides in Action

Now let us look at these overrides in action. Our implementation includes the following:

- an es-mx Locale
- a Central American Marketing Community
- a Brief Content Type
- an Article Workflow
- An Auto Translation configuration for the Brief Content Type in the Central American Marketing Community:

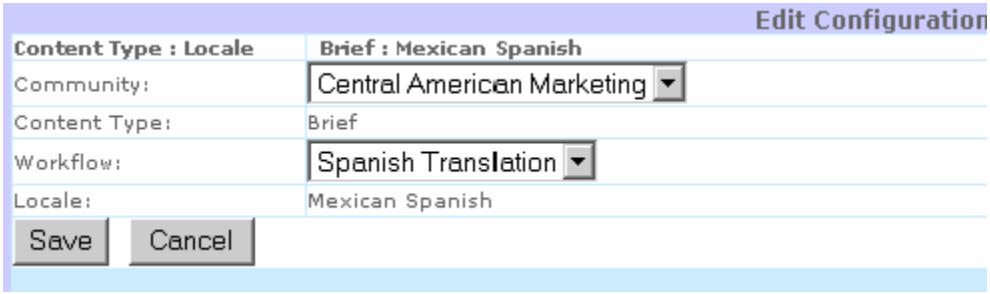

Figure 31: Example Auto Translation Configuration for the Brief ContentType in the Central American Markeing Community

- In the Article Workflow, the Approve Transition from QA to Public includes the sys_createTranslation Workflow Action:

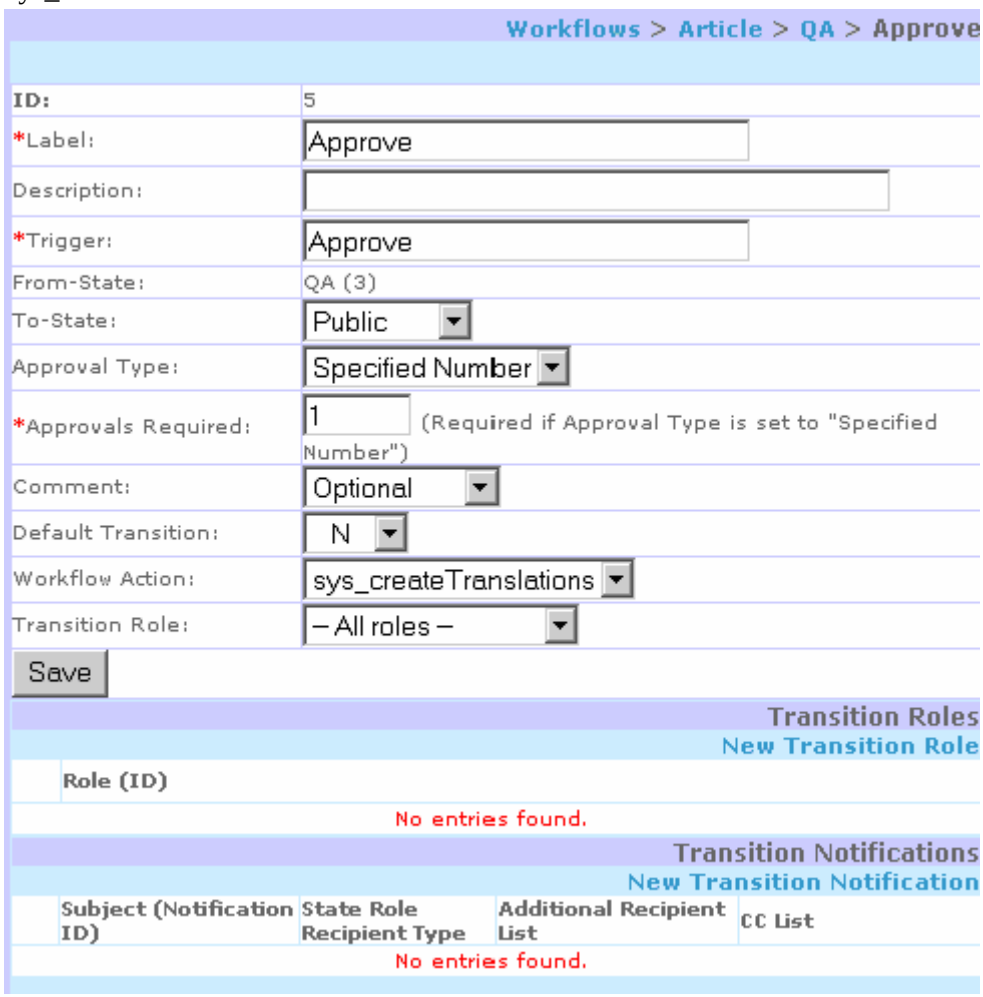

Figure 32: Approve Transition configured with the sys_createTranslation Workflow Action

Now, let us create a Brief Content Item:

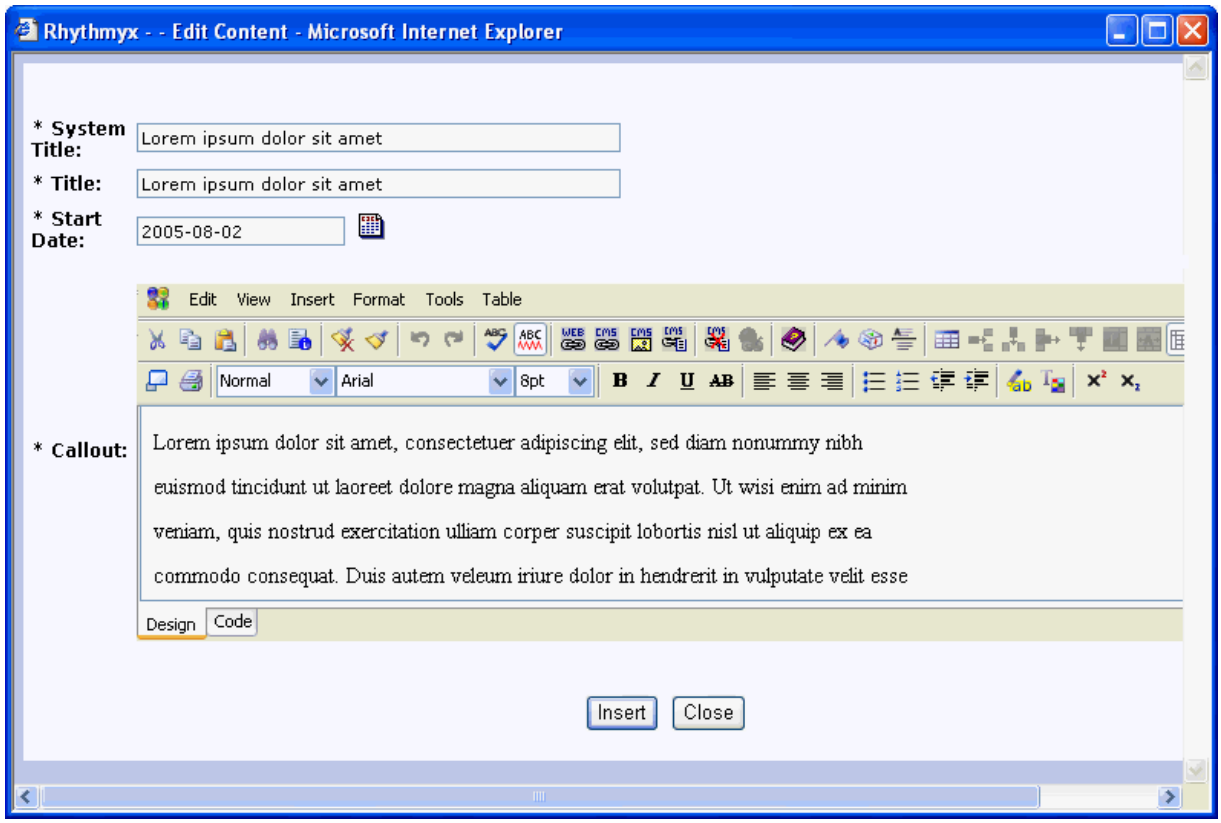

Figure 33: Example Brief Content Item

The following graphic shows this Content Item in Content Explorer:

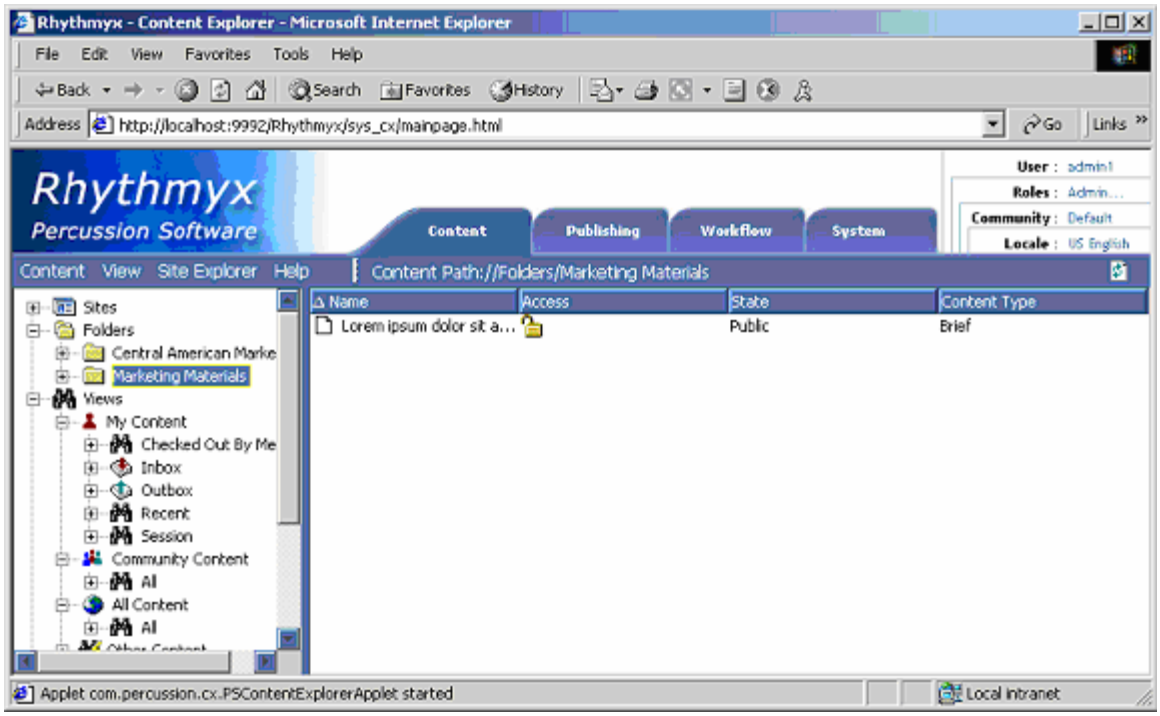

Figure 34: Example Brief Content Item in Content Explorer

Note the Community, Workflow, and Locale in the Properties of this Content Item.

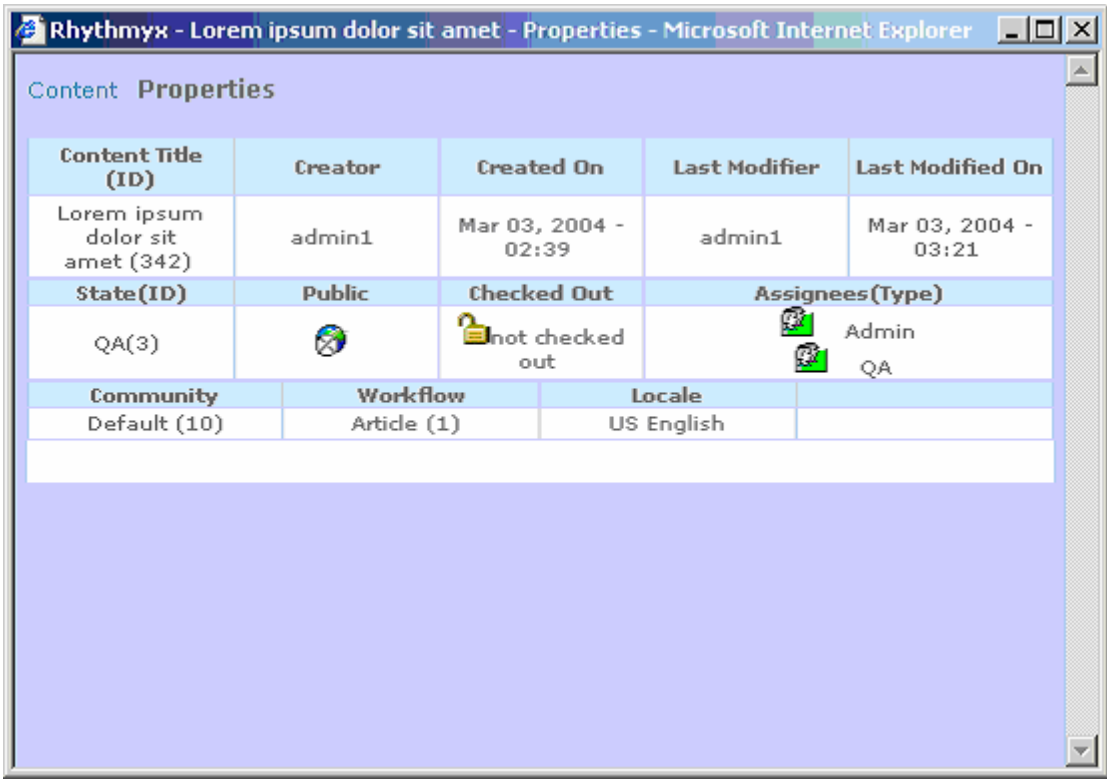

Figure 35: Properties of the example Brief Content Item

When we Transition this Content Item to Public, a Translation Item will be created automatically. It will have the name "[es-mx] Translation of Lorem ipsom dolor sit amet". (Note that the Translation is automatically added to the same folder at the Owner Content Item; to add it to a different folder, you would need to write a new Effect for the Translation Relationship that would add it to a different Folder.)

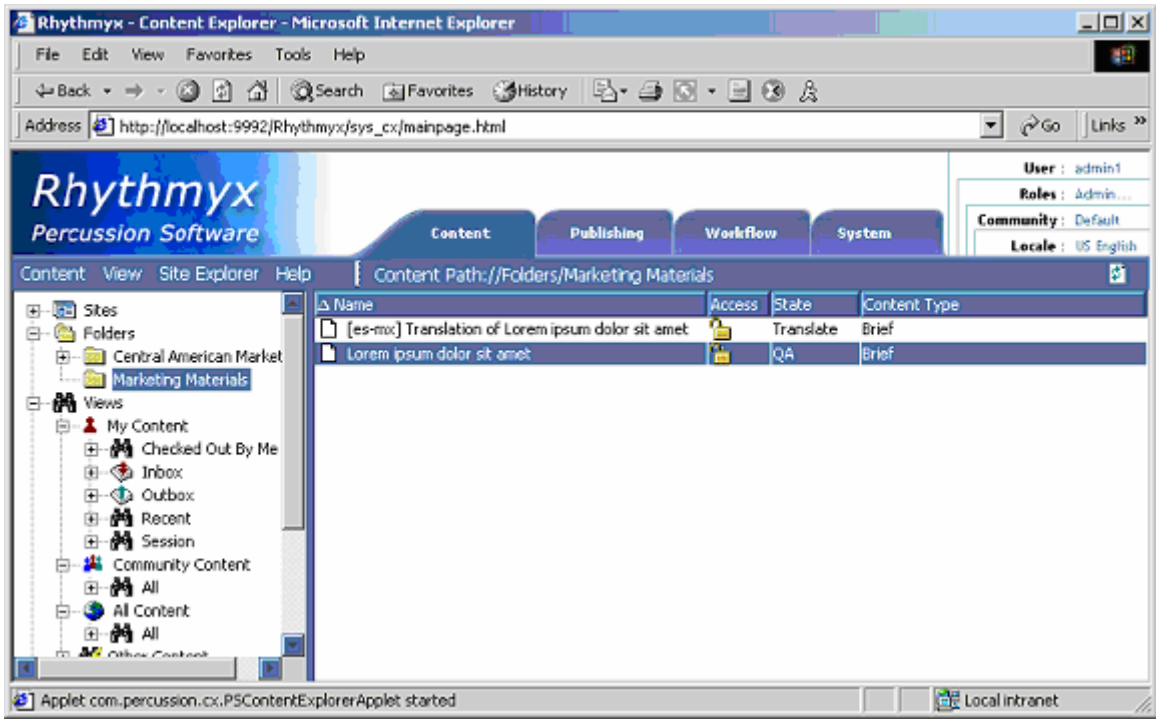

Figure 36: es-mx Translation of the example Content Item

Note that the Community, Workflow, and Locale have changed in the Translation Item:

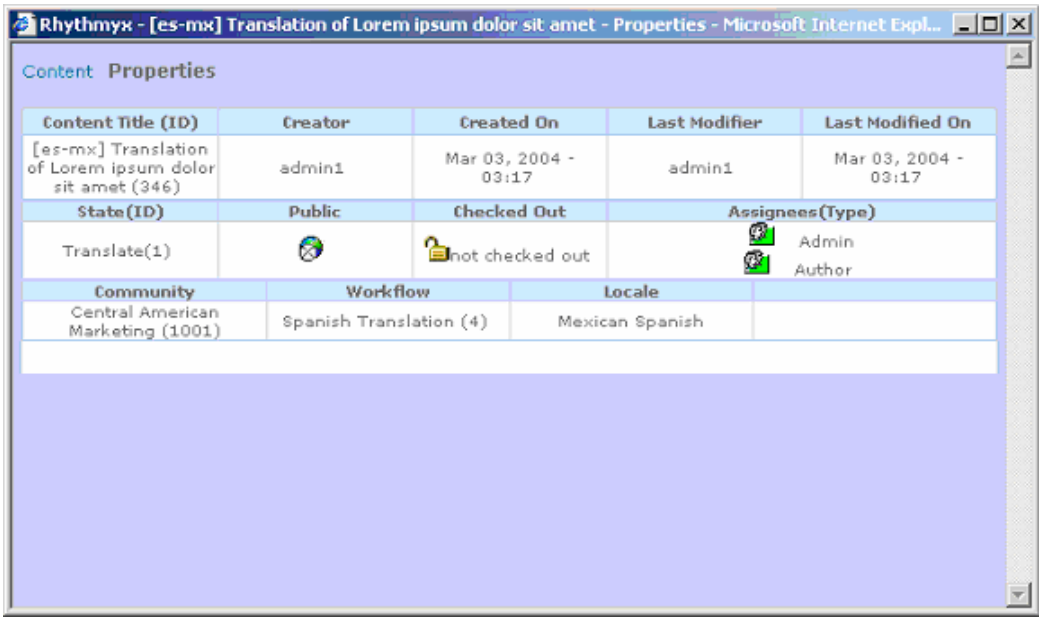

Figure 37: Properties of the es-mx Translation of the example Content Item

CHAPTER 6

Writing Effects

Effects are Rhythmyx server extensions used to extend the Relationship engine. Effects are extensions of the IPSExtension Java extension. Thus, Effects must follow all of the requirements of Rhythmyx extensions, including thread safety. (For details about thread safety, see any standard Java reference.) Effects must implement the IPSEffect interface. To make an Effect available in the system you must register it. (For details about registering extensions see "Adding a New Java Extension" in the Rhythmyx Server Administrator online Help.). When registering an Effect, you must select com.percussion.extension.IPSEffect as the Supported Interface.

Example Effect

The code of the default PSValidateFolder Effect provided by Percussion Software is provided as an example of an Effect.

```
/**[ PSValidateFolder.java
]************************************************
 *
 * COPYRIGHT (c) 2002 by Percussion Software, Inc., Stoneham, MA USA.
 * All rights reserved. This material contains unpublished, copyrighted
 * work including confidential and proprietary information of
Percussion.
 *

*********************************************************************
******/
package com.percussion.relationship.effect;

import com.percussion.cms.IPSCmsErrors;
import com.percussion.cms.PSCmsException;
import com.percussion.cms.objectstore.IPSComponentProcessor;
import com.percussion.cms.objectstore.IPSRelationshipProcessor;
import com.percussion.cms.objectstore.PSComponentProcessorProxy;
import com.percussion.cms.objectstore.PSComponentSummaries;
import com.percussion.cms.objectstore.PSComponentSummary;
import com.percussion.cms.objectstore.PSDbComponent;
import com.percussion.cms.objectstore.PSKey;
import com.percussion.cms.objectstore.PSProcessorProxy;
import com.percussion.cms.objectstore.PSRelationshipFilter;
import com.percussion.cms.objectstore.PSRelationshipProcessorProxy;
import com.percussion.design.objectstore.PSLocator;
import com.percussion.design.objectstore.PSRelationship;
import com.percussion.design.objectstore.PSRelationshipConfig;
import com.percussion.error.PSException;
import com.percussion.extension.PSExtensionProcessingException;
import com.percussion.extension.PSParameterMismatchException;
import com.percussion.relationship.IPSExecutionContext;
import com.percussion.relationship.PSAttemptResult;
import com.percussion.relationship.PSEffect;
import com.percussion.relationship.PSEffectResult;
import com.percussion.server.IPSRequestContext;
import com.percussion.server.PSRequest;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import org.w3c.dom.Element;

/**
 * This effect is aimed at validating a new folder based on the rules
outlined
 * below:
```

```
 * <p>
 * The owner of the relationship must be a folder and the dependent's
name
 * must be different (case-insensitive) than the names of all other
children
 * of the owner unless it is the exact same objects. A folder does not
allow
 * children with duplicate names. If any validation fails, an exception
that
 * terminates the processing is thrown. The effect will return
immediately for
 * any context except RS_CONSTRUCTION.
 * <p>
 * This effect does not need any parameters.
 */
public class PSValidateFolder extends PSEffect
{
   /**
    * Override the methode in the base class. This effect is meant to be
run
    * RS_CONSTRUCTION context and hence will return <code>false</code>
for all
    * other contexts.
    */
   public void test(Object[] params, IPSRequestContext request,
      IPSExecutionContext context, PSEffectResult result)
   {

      if (!context.isConstruction())
      {
         result.setWarning("Illegal Context, expected to be
construction.");
         return;
      }

      PSRelationship originatingRel =
context.getOriginatingRelationship();

      if (originatingRel == null
         || !originatingRel.getConfig().getName().equals(
            PSRelationshipConfig.TYPE_FOLDER_CONTENT))
      {
         result.setWarning(
            "The originating relationship is not of type '"
               + PSRelationshipConfig.TYPE_FOLDER_CONTENT
               + "'.");
         return;
      }

      Set relsProcessed = (Set)m_tlRelationshipsProcessed.get();
      if(relsProcessed==null)
      {
         relsProcessed = new HashSet();
         m_tlRelationshipsProcessed.set(relsProcessed);
      }
```

```
      /*
       * Folder validation needs to be done only once
       */
      if(relsProcessed.contains(originatingRel))
      {
         result.setWarning("The relationship is already processed");
         return;
      }

      try
      {
         PSLocator depLocator = originatingRel.getDependent();
         PSLocator ownerLocator = originatingRel.getOwner();

         validateUniqueDepName(ownerLocator, depLocator, null, request,
result);
      }
      finally
      {
         relsProcessed.add(originatingRel);
      }
   }

   /**
    * Validates the supplied dependent with the following rules:
    * <p>
    * The dependent's name must be different (case-insensitive) then
    * the names of all other children of the owner (folder) unless the
same
    * object is already a child of the supplied owner. A folder does not
allow
    * children with duplicate names.
    *
    * @param owner the locator of the owner folder, not
<code>null</code>.
    * @param dependent the locator of the dependent item, not
    *    <code>null</code>.
    * @param depName The name (or sys_title) of the dependent. It may be
    *    <code>null</code> or empty in which case it will be looked up
using
    *    the supplied dependent locator.
    * @param request the current request, not <code>null</code>.
    * @return <code>true</code> if validated, <code>false</code>
otherwise.
    */
   public static boolean validateUniqueDepName(PSLocator owner,
      PSLocator dependent, String depName, PSRequest request)
   {
      if (owner == null)
         throw new IllegalArgumentException("owner may not be null");

      if (dependent == null)
         throw new IllegalArgumentException("dependent may not be
null");
```

```
      if (request == null)
         throw new IllegalArgumentException("request may not be null");

      PSAttemptResult result = new PSAttemptResult();
      validateUniqueDepName(owner, dependent, depName, request, result);

      return (result.getException() == null);
   }

   /**
    * See {@link validateUniqueDepName(PSLocator, PSLocator, String,
PSRequest)}
    * for description.
    *
    * @param ownerLocator the locator of the owner folder, not
    *    <code>null</code>.
    * @param depLocator the locator of the dependent item, not
    *    <code>null</code>.
    * @param depName The name (or sys_title) of the dependent. It may be
    *    <code>null</code> or empty in which case it will be looked up
using
    *    the supplied dependent locator.
    * @param result the result object into which the result of the
validation
    *    will be set, assume it is not <code>null</code>.
    */
   private static void validateUniqueDepName(PSLocator ownerLocator,
      PSLocator depLocator, String depName, Object request,
      PSEffectResult result)
   {
      try
      {
         IPSRelationshipProcessor relProxy = new
PSRelationshipProcessorProxy(
            PSProcessorProxy.PROCTYPE_SERVERLOCAL, request);

         PSRelationshipFilter filter = new PSRelationshipFilter();
         filter.setOwner(ownerLocator);
         PSComponentSummaries children = relProxy.getSummaries(filter,
false);

         if (children.isEmpty())
         {
            result.setSuccess();
            return;
         }

         // need to lookup the name if it was not supplied
         if (depName == null || depName.trim().length() == 0)
         {
            PSComponentSummary depSummary = getSummary(depLocator,
request);
            depName = depSummary.getName();
         }
```

```
            Iterator walker = children.iterator();
            while (walker.hasNext())
            {
               PSComponentSummary childSummary =
                  (PSComponentSummary) walker.next();

               if (childSummary.getName().equalsIgnoreCase(depName))
               {
                  /*
                   * This is only an error if it is not the same item
   (meaning the
                   * same content id). The revision is not considered for
   this
                   * test.
                   */
                  if (childSummary.getCurrentLocator().getId() !=
                     depLocator.getId())
                  {
                     PSComponentSummary summary = getSummary(ownerLocator,
                        request);
                     String parentName = summary.getName();
                     Object[] args =
                     {
                        depName,
                        parentName,
                        String.valueOf(depLocator.getId()),
                        String.valueOf(depLocator.getRevision())
                     };
                     PSCmsException exception = new PSCmsException(
                        IPSCmsErrors.FOLDER_REL_ERROR_DUPLICATED_CHILDNAME,
   args);

                     result.setError(exception);
                     result.setKeys(new PSKey[] { depLocator });

                     return;
                  }
               }
            }

            result.setSuccess();
         }
         catch (PSException ex)
         {
            result.setError(ex);
         }
      }

      /**
       * Get the summary information for the supplied locator.
       *
       * @param locator The locator, assume not <code>null</code>.
       *
       * @param request The current request, assume not <code>null</code>.
       *
       * @return The summary info, never <code>null</code>.
```

```
     *
     * @throws PSException if an error occurs while retrieving the
summary info.
     */
   private static PSComponentSummary getSummary(
      PSLocator locator,
      Object request) throws PSException
   {
      PSComponentSummary summary = null;

      IPSComponentProcessor compProxy =
         new PSComponentProcessorProxy(
            PSProcessorProxy.PROCTYPE_SERVERLOCAL,
            request);
      Element[] summaries =
         compProxy.load(
            PSDbComponent.getComponentType(PSComponentSummaries.class),
            new PSKey[] { locator });
      if (summaries == null || summaries.length < 1)
      {
         Object[] args =
            {
                String.valueOf(locator.getId()),
                String.valueOf(locator.getRevision())
            };
         throw new PSCmsException(IPSCmsErrors.FAILED_GET_SUMMARY,
args);
      }

      PSComponentSummaries depSummaries =
         new PSComponentSummaries(summaries);
      summary = (PSComponentSummary)depSummaries.iterator().next();

      return summary;
   }

   //Implementation of the interface method
   public void attempt(Object[] params, IPSRequestContext request,
      IPSExecutionContext context, PSEffectResult result)
      throws PSExtensionProcessingException,
PSParameterMismatchException
   {
      //Folder validation effect does not do any special processing
      result.setSuccess();
   }

   //Implementation of the interface method
   public void recover(Object[] params, IPSRequestContext request,
      IPSExecutionContext context, PSExtensionProcessingException e,
      PSEffectResult result)
      throws PSExtensionProcessingException
   {
      //Folder validation effect does not need to recover anything
      result.setSuccess();
   }
```

```
   /**
    * Thread local storage of the processed relationship. This is just
to
    * avoid unnecessary processing of the same relationship for each
current
    * relationship. We need to validate the originating folder
relationship
    * only once not while processing each relationship around the
original
    * owner item.
    */
   private static ThreadLocal m_tlRelationshipsProcessed = new
ThreadLocal();
}
```

CHAPTER 7

Default Relationships

Percussion Software provides the following default Relationships with Rhythmyx:

- Related Content
- Dependent Related Content
- Translation
- Dependent Translation
- New Copy
- Promotable Version
- Folder Content

Active Assembly

The Active Assembly Relationship creates a simple association between the owner and the dependent, including Inline Links and Inline Images. User's create an Active Assembly Relationship whenever they use Active Assembly to associate one Content Item with another.

General Properties

Name - *Active Assembly*. Non-editable.

Label - *Active Assembly*. Non-editable.

Category - *Active Assembly*. Non-editable.

System Properties: (all may be edited)

rs_useownerrevision - Specifies whether to use the owner's revision ID as part of the owner locator key. *Yes*. *Locked*.

rs_usedependentrevision - Specifies whether to use the dependent's revision ID as part of the dependent locator key. *No*. *Locked*.

rs_expirationtime - If this relationship can expire, this property specifies the expiration date/time. If this relationship cannot expire, this property is set to null. *Null* (relationship cannot expire).

rs_useserverid - Specifies whether to use the server ID (rxserver) for executing effects.. If set to *No*, the current user is used instead of the server ID. *Yes*.

rs_islocaldependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.) *Yes.*

rs_skippromotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed. *No*

User Properties (all may be edited) User property values are always filled at runtime. Note that any custom Relationship in the Active Assembly Category must include these User Properties.

sys_slotid - Slot ID in which relationship is used. *None.*

sys_sortrank - Sort rank within Slot. *1*.

sys_variantid - Variant ID. *None.*

rs_inlinerelationship - specifies that the Active Assembly Relationship

Cloning Properties (all may be edited)

Allow Cloning - Whether or not relationship may be cloned. *Yes.*

Locked - Whether local processing can override cloning properties. *Yes.*

rs_cloneshallow - Whether or not shallow clones of the Relationship may be created. *Yes, if activating Relationship Category is Promotable Version or New Copy; otherwise, no.*

rs_clonedeep - Whether or not deep clones of the Relationship may be created. *Yes if activating Relationship Category is Translation; otherwise, No.*

Clone Field Overrides None

Request Pre-processing (Pre Exits)

None. Any number may be added.

Result Document Processing (Post Exits)

None. Any number may be added.

Effects

None.

Active Assembly - Mandatory

Slight modification to the Active Assembly Relationship that prevents a Content Item from going Public unless all descendants related through this Relationship Type are Public, or can go Public. Use this Relationship when you want to ensure that a Content Item cannot go Public unless the Dependent Content Item in the Active Assembly is also Public.

General Properties

Name - *Active Assembly - Mandatory*. Non-Editable.

Label - *Active Assembly - Mandatory*. Non-Editable.

Category - *Active Assembly*.

System Properties: (all may be edited)

rs_useownerrevision - Specifies whether to use the owner's revision ID as part of the owner locator key. *Yes*. *Locked*.

rs_usedependentrevision - Specifies whether to use the dependent revision ID as part of the dependent locator key. *No*. *Locked*.

rs_expirationdate - If this relationship can expire, this property specifies the expiration date/time. If this relationship cannot expire, this property is set to null. *Null* (relationship cannot expire).

rs_useserverid - Specifies whether to use the server ID (rxserver) for executing effects.. If set to *No*, the current user is used instead of the server ID. *Yes*.

rs_islocaldependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.) *Yes.*

rs_skippromotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed. *No*

User Properties (all may be edited) User property values are always filled at runtime. Note that any custom Relationship in the Active Assembly Category must include these User Properties.

sys_slotid - Slot ID in which relationship is used. *None.*

sys_sortrank - Sort rank within Slot. *1*.

sys_variantid - Variant ID. *None.*

Cloning Properties (all may be edited)

Allow Cloning - Whether or not relationship may be cloned. *Yes.*

Locked - Whether local processing can override cloning properties. *Yes.*

rs_cloneshallow - Whether or not shallow clones of the Relationship may be created. *Yes, if activating Relationship Category is Promotable Version or New Copy; otherwise, no.*

rs_clonedeep - Whether or not deep clones of the Relationship may be created. *Yes if activating Relationship Category is Translation; otherwise, No.*

Clone Field Overrides None

Request Pre-processing (Pre Exits)

None. Any number may be added.

Result Document Processing (Post Exits)

None. Any number may be added.

Effects

sys_PublishMandatory: Direction: *Down*

forceTransition: *No*

ownerTransitionName: *Null*

dependentTransitionName: *Null*

sys_UnpublishMandatory: Direction: *Down*

forceTransition: *No*

ownerTransitionName: *Null*

dependentTransitionName: *Null*

Folder Content

The Folder Relationship associates a Folder with a Content Item in the Folder. Rhythmyx creates an instance of this Relationship type whenever a user adds a Content Item to a Folder.

General Properties

Name - *Folder Content.* Non-editable.

Label - *Folder Content.* Non-editable.

Category - *Folder*. Non-editable.

System Properties: (all may be edited)

rs_useownerrevision - Specifies whether to use the owner revision ID as part of the owner locator key. *No. Locked.*

rs_usedependentrevision - Specifies whether to use the dependent revision ID as part of the dependent locator key. *No*. *Locked*.

rs_expirationdate - If this relationship can expire, this property specifies the expiration date/time. If this relationship cannot expire, this property is set to null. *Null* (relationship cannot expire).

rs_useserverid - Specifies whether to use the server ID (rxserver) for executing effects.. If set to *No*, the current user is used instead of the server ID. *Yes*.

rs_islocaldependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.) *No.*

rs_usecommunityfilter - Specifies whether Dependent Content Items available to the Relationship are filtered based on the Community of the user logged in to the system. *Yes*. *Locked*.

rs_skippromotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed. *Yes*.

User Properties User property values are always filled at runtime.

None. Any number may be added.

Cloning Properties (all may be edited)

Allow Cloning - Whether or not relationship may be cloned. *Yes.*

Locked - Whether local processing can override cloning properties. *Yes.*

rs_cloneshallow - Whether or not shallow clones of Relationships may be created. *No.*

rs_clonedeep - Whether or not deep clones of Relationships may be created. *Yes when Activating Relationship Name is "Translation - Mandatory"; otherwise, no.*

Clone Field Overrides None

Request Pre-processing (Pre-Exits)

None. Any number may be added.

Result Document Processing (Post-Exits)

None. Any number may be added.

Effects

sys_TouchParentFolderEffect: Direction: *Down*

rxs_NavFolderEffect: Direction: *Either* Conditions:
PSXSingleHtmlParameter/rxs_disableNavFolderEffect=y

New Copy

A New Copy Relationship creates an association between an owner Content Item and its clone. The primary purpose of this Relationship is to ensure that no more than one New Copy clone of a Content Item exists. Use this Relationship when you want to have a clone of a Content Item that co-exists with the original Content Item (rather than superseding it when reaching Public, as occurs using the Promotable Version Relationship). Rhythmyx creates an instances of this Relationship type whenever a user creates a new Copy of a Content Item (in other words, when the user chooses *Create > New Copy* from an Action Menu).

General Properties

Name - *New Copy*. Non-editable.

Label - *New Copy*. Non-editable.

Category - *New Copy*. Non-editable.

System Properties: (all may be edited)

rs_useownerrevision - Specifies whether to use the owner's revision ID as part of the owner locator key. *Yes*. *Locked*.

rs_usedependentrevision - Specifies whether to use the dependent's revision ID as part of the dependent locator key. *No*. *Locked*.

rs_expirationtime - If this relationship can expire, this property specifies the expiration date/time. If this relationship cannot expire, this property is set to null. *Null* (relationship cannot expire).

rs_useserverid - Specifies whether to use the server ID (rxserver) for executing effects.. If set to *No*, the current user is used instead of the server ID. *Yes*.

rs_islocaldependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.) *No.*

rs_skippromotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed. No

User Properties User property values are always filled at runtime.

None. Any number may be added.

Cloning Properties (all may be edited)

Allow Cloning - *No.*

Locked - Whether local processing can override cloning properties. *Yes.*

rs_cloneshallow - Whether or not shallow clones of relationship may be created. *Not applicable*.

rs_clonedeep - Whether or not deep clones of relationship may be created. *Not applicable*.

Clone Field Overrides

Field	UDF	Conditions
sys_title	sys_CloneTitle	None

Parameters

Name	Value
Pattern	Copy ($clone_count) of {0}
InsertionItem0	PSXContentItemStatus/ CONTENTSTATUS.TITLE

sys_communityid	sys_Literal	None

Parameters

Name	Value
Default	PSXContentItemStatus/ CONTENTSTATUS.COMMUNITYID
overrideParameterName	sys_communityid_override

sys_workflowid	sys_Literal	None

Parameters

Name	Value
Default	PSXContentItemStatus/WORKFLOW APPS.WORKFLOWAPPID
overrideParameterName	sys_workflowid_override

Request Pre-processing (Pre-Exits)

None. Any number may be added.

Result Document Processing (Post-Exits)

None. Any number may be added.

Effects

sys_AddCloneToFolder

Direction: *Down*

Promotable Version

A Promotable Version Relationship creates an association between an owner Content Item and its clone, which specifies that when the clone reaches the public State, the owner is transitioned to the archive State. Use this Relationship when you want a clone of a Content Item to supersede the original Content Item when the clone goes Public. Rhythmyx creates an instance of this Relationship Type when a user chooses *Create > New Version* from an Action Menu.

General Properties

Name - *Promotable Version.* Non-editable.

Label - *Promotable Version.* Non-editable.

Category - *Promotable Version*. Non-editable.

System Properties: (all may be edited)

rs_useownerrevision - Specifies whether to use the owner's revision ID as part of the owner locator key. *Yes*. *Locked*.

rs_usedependentrevision - Specifies whether to use the dependent's revision ID as part of the dependent locator key. *No*. *Locked*.

rs_expirationdate - If this relationship can expire, this property specifies the expiration date/time. If this relationship cannot expire, this property is set to null. *Null* (relationship cannot expire).

rs_useserverid - Specifies whether to use the server ID (rxserver) for executing effects.. If set to *No,* the current user is used instead of the server ID. *Yes*.

rs_islocaldependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.) *No.*

rs_skippromotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed. No

User Properties User property values are always filled at runtime.

None. Any number may be added.

Cloning Properties (all may be edited)

Allow Cloning - *No.*

Locked - Whether local processing can override cloning properties. *Yes.*

rs_cloneshallow - Whether or not shallow clones of the Relationship may be created. *Not Applicable.*

rs_clonedeep - Whether or not deep clones of the Relationship may be created. *Not Applicable.*

Clone Field Overrides

Field	UDF	Conditions
sys_title	sys_CloneTitle	None

Parameters

Name	Value
Pattern	PV Copy ($clone_count) of {0}
InsertionItem0	PSXContentItemStatus/ CONTENTSTATUS.TITLE

sys_communityid	sys_Literal	None

Parameters

Name	Value
p1	PSXContentItemStatus/CONTENT STATUS.COMMUNITYID

sys_workflowid	sys_Literal	None

Parameters

Name	Value
p1	PSXContentItemStatus/WORKFLOW APPS.WORKFLOWAPPID

Request Pre-processing (Pre-Exits)

None. Any number may be added.

Result Document Processing (Post-Exits)

None. Any number may be added.

Effects

sys_Promote

Direction: *Up*.

transitionName *Null*

sys_AddCloneToFolder

Direction: *Down*

Translation

The Translation Relationship associates an owner Content Item with a clone of itself. When Rhythmyx clones the owner, it also clones all of its dependents, and they form Translation Relationships with the owner dependents. Use this Relationship when you want to create a clone that will be translated to another language and will be published independent of the Owner Content Item.

General Properties

Name - *Translation.* Non-editable.

Label - *Translation.* Non-editable.

Category - *Translation.* Non-editable.

System Properties: (all may be edited)

rs_useownerrevision - Specifies whether to use the owner's revision ID as part of the owner locator key. *Yes*. *Locked*.

rs_usedependentrevision - Specifies whether to use the dependent's revision ID as part of the dependent locator key. *No*. *Locked*.

rs_expirationdate - If this relationship can expire, this property specifies the expiration date/time. If this relationship cannot expire, this property is set to null. *Null* (relationship cannot expire).

rs_useserverid - Specifies whether to use the server ID (rxserver) for executing effects.. If set to *No*, the current user is used instead of the server ID. *Yes*.

rs_islocaldependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.) *No.*

rs_skippromotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed. No

User Properties User property values are always filled at runtime.

None. Any number may be added.

Cloning Properties (all may be edited)

Allow Cloning - Whether or not relationship may be cloned. *No.*

Locked - Whether local processing can override cloning properties. *Yes.*

rs_cloneshallow - Whether or not shallow clones of the Relationship may be created. *Not applicable.*

rs_clonedeep - Whether or not deep clones of the Relationship may be created. *Not applicable.*

Clone Field Overrides

Field	UDF	Conditions
sys_title	sys_CloneTitle	None

Parameters

Name	Value
Pattern	[{0}] Copy of {1}
InsertionItem0	HTMLSingleParameter/sys_lang
InsertionItem1	PSXContentItemStatus/ CONTENTSTATUS.TITLE

sys_communityid	sys_CloneFieldOverride	None

Parameters

Name	Value
Url	../sys_trFieldOverride/ TranslationFieldOverride.xml
FieldElemName	CommunityId
ParamName1	sys_contentid
ParamValue1	PSXContentItemStatus/ CONTENTSTATUS. CONTENTID
ParamName2	sys_lang
ParamValue2	HTMLSingleParameter/ sys_lang

Field	UDF	Conditions
sys_workflowid	sys_CloneFieldOverride	None

Parameters

Name	Value
Url	../sys_trFieldOverride/ TranslationFieldOverride.xml
FieldElemName	WorkflowId
ParamName1	sys_contentid
ParamValue1	PSXContentItemStatus/ CONTENTSTATUS. CONTENTID
ParamName2	sys_lang
ParamValue2	HTMLSingleParameter/ sys_lang

sys_lang	sys_Literal	None

Parameters

Name	Value
p1	HTMLSingleParameter/sys_lang

Request Pre-processing (Pre-Exits)

sys_TranslationConstraint - This Exit runs before creating a new Translation Copy. It prevents creation of multiple Translation Copies of the owner in the same Language.

Result Document Processing (Post-Exits)

None. Any number may be added.

Effects

None.

Translation - Mandatory

Slight modification to the Translation Relationship that prevents a Content Item from going Public unless all descendants related through this Relationship Type are Public, or can go Public. Use his Relationship when you want to create a clone of a Content Item that will be translated and you want to prevent the original Content Item from going Public until the clone is also ready to go Public.

General Properties

Name - *Translation - Mandatory.* Non-editable.

Label - *Translation - Mandatory.* Non-editable.

Category - *Translation.* Non-editable.

System Properties: (all may be edited)

rs_useownerrevision - Specifies whether to use the owner's revision ID as part of the owner locator key. *Yes*. *Locked*.

rs_usedependentrevision - Specifies whether to use the dependent's revision ID as part of the dependent locator key. *No*. *Locked*.

rs_expirationdate - If this relationship can expire, this property specifies the expiration date/time. If this relationship cannot expire, this property is set to null. *Null* (relationship cannot expire).

rs_useserverid - Specifies whether to use the server ID (rxserver) for executing effects.. If set to *No*, the current user is used instead of the server ID. *Yes*.

rs_islocaldependency - Specifies whether the Multi-Server Manager should treat the dependent as a local dependency. (See the Rhythmyx Multi-Server Manager documentation for more information.) *No.*

rs_skippromotion - Specifies whether to repoint the Relationship to the depended object in a Promotable Version Relationship when the depended it promoted to Public. If checked, the Relationship is not repointed. If unchecked, the Relationship is repointed. No

User Properties User property values are always filled at runtime.

None. Any number may be added.

Cloning Properties (all may be edited)

Allow Cloning - Whether or not relationship may be cloned. *Yes*.

Locked - Whether local processing can override cloning properties. *Yes*.

rs_cloneshallow - Whether or not shallow clones of the Relationship may be created. *No.*

rs_clonedeep - Whether or not deep clones of the Relationship may be created. *Yes if the activating Relationship Category is Promotable Version; otherwise, no*.

Clone Field Overrides

Field	UDF	Conditions
sys_title	sys_CloneTitle	None

Parameters

Name	Value
Pattern	[{0}] Copy of {1}
InsertionItem0	HTMLSingleParameter/sys_lang
InsertionItem1	PSXContentItemStatus/ CONTENTSTATUS.TITLE

sys_communityid	sys_CloneFieldOverride	None

Parameters

Name	Value
Url	../sys_trFieldOverride/ TranslationFieldOverride.xml
FieldElemName	CommunityId
ParamName1	sys_contentid
ParamValue1	PSXContentItemStatus/ CONTENTSTATUS. CONTENTID
ParamName2	sys_lang
ParamValue2	HTMLSingleParameter/ sys_lang

Field	UDF	Conditions
sys_workflowid	sys_CloneFieldOverride	None

Parameters

Name	Value
Url	../sys_trFieldOverride/ TranslationFieldOverride.xml
FieldElemName	WorkflowId
ParamName1	sys_contentid
ParamValue1	PSXContentItemStatus/ CONTENTSTATUS. CONTENTID
ParamName2	sys_lang
ParamValue2	HTMLSingleParameter/ sys_lang

sys_lang	sys_Literal	None

Parameters

Name	Value
p1	HTMLSingleParameter/sys_lang

Request Pre-processing (Pre-Exits)

sys_TranslationConstraint - This Exit runs before creating a new Translation Copy. It prevents creation of multiple Translation Copies of the owner in the same Language.

Additional pre-exits may be added.

Result Document Processing (Post-Exits)

None. Any number may be added.

Effects

sys_PublishMandatory: Direction: *Up*

forceTransition: *No*

ownerTransitionName: *Null*

dependentTransitionName: *Null*

sys_UnpublishMandatory: Direction: *Up*

forceTransition: *No*

ownerTransitionName: *Null*

dependentTransitionName: *Null*

CHAPTER 8

Default Effects

Percussion Software provides the following default Effects with Rhythmyx:

- ***sys_AddCloneToFolder*** (on page 92)
- ***sys_isCloneExists*** (on page 93)
- ***sys_Notify*** (on page 94)
- ***sys_Promote*** (on page 95)
- ***sys_PublishMandatory*** (on page 96)
- ***sys_TouchParentFolderEffect*** (on page 99)
- ***sys_UnpublishMandatory*** (see "sys_PublishMandatory" on page 96)
- ***sys_Validate*** (on page 101)
- ***sys_ValidateFolder*** (on page 102)
- ***rxs_NavFolderEffect*** (on page 90)
- ***rxs_NavFolderCache*** (see "rxs_NavFolderEffect" on page 90)

rxs_NavFolderEffect

This effect is used in Rhythmyx internal implementations.

rxs_NavFolderCache

This effect is used in Rhythmyx internal implementations.

sys_AddCloneToFolder

This Effect associates a new clone of a Content Item with the Folder or Folders in which the Owner in the Relationship resides. The Effect runs when:

- The context is Relationship creation;
- The request includes the HTML parameter sys_folderid, with a value that is not null and not empty (if the request includes multiple values for this parameter, the clone will be added to each specified Folder); and
- The Relationship that is being processed is the Relationship that originally created the clone. (This requirement ensures that Effect will only be processed once in the life of the Relationship.)

This Effect is assigned to Relationships that allow users to create clones directly in the Content Explorer interface, such as the New Copy and Promotable Version Relationships.

Note that the Effect does not check whether the clone is already associated with any of the specified Folders.

sys_isCloneExists

This Effect prevents cloning of a Translation Relationship if a Translation Relationship already exists to a Content Item in the target Locale.

sys_Notify

Generates a Notification to all assignees for items that match the parameters of the Effect.

Parameters

Name	Description	Required?
workflowid	The ID of the Workflow for which to generate the Notification	Yes
stateid	The ID of the Workflow State for which to generate the Notification	Yes
transitionid	The ID of the Transition for which to generate the Notification	Yes
username	The username for which to generate the Notification	Yes

sys_Promote

When the Content Item enters a Public State the first time, it replaces the other Content Item in the Relationship. To execute the replacement:

- Transitions the original Content Item, using either the Transition specified in the transitionName parameter or the default Transition. This Transition should move the original Content Item to an Archive State.
- Updates all Relationships that specified the original Content Item as the Dependent to specify the newly-promoted Content Item as the Dependent.
- Removes all Clonable Relationships from the original Content Item
- Updates all other Relationships that specified the original Content Item as the Owner to specify the newly-promoted Content Item as the Owner.

Parameters

Name	Description	Required?
transitionName	Internal name (value of the Trigger field of the Edit Transition page) of the Transition to use to Transition the Content Item to another State. If no value is provided for this parameter, the first Transition (alphabetically by the value of the Trigger field) in the State for which the value of the Default property is "yes" is used.	No

sys_PublishMandatory

This Effect is used in processing mandatory Relationships. It prevents Transition of the Content Item that activated the Effect to a Public State unless the other Content Item in the Relationship is already in a Public State.

If the value of the forceTransition parameter is "yes", and either a Transition for which the value of the Trigger field matches the value of the appropriate TransitionName parameter or a Default Transition is specified from the current State of the associated Content Item to the Public State of that Content Item's Workflow, the associated Content Item will be Transitioned to the Public State along with the original Content Item. If the associated Content Item cannot be Transitioned to a Public State, Rhythmyx generates an error and the original Content Item is not allowed to make the Transition. The following flowchart illustrates the processing of this Effect:

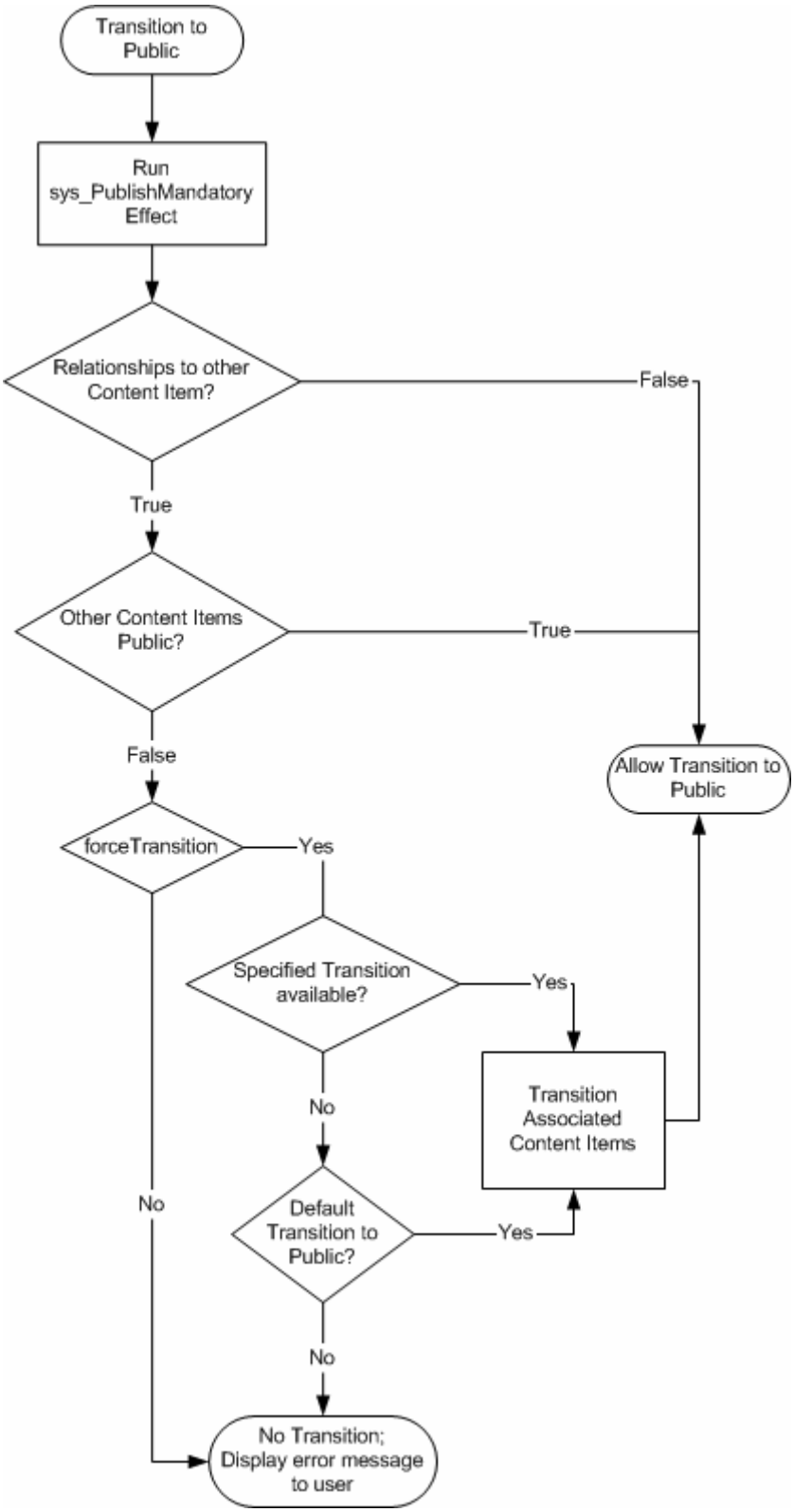

Figure 38: Processing of the sys_PublishMandatory Effect

Parameters

Name	Description	Required?
forceTransition	Boolean flag ("yes" or "no") that specifies whether the Content Item associated in the Relationship will be forced to make the Transition, if the Transition is possible. If the value of this parameter is "no" and the associated Content Item is already in a Public State, the Transition of the original Content Item fails.	Yes
ownerTransitionName	Internal name (value of the Trigger field of the Edit Transition page) of the Transition to use to Transition the associated Content Item to another State if that Content Item is the Owner in the Relationship. If no value is provided for this parameter, the first Transition (alphabetically by the value of the Trigger field) in the State for which the value of the Default property is "yes" is used.	No
dependentTransitionName	Internal name (value of the Trigger field of the Edit Transition page) of the Transition to use to Transition the associated Content Item to another State if that Content Item is the Dependent in the Relationship. If no value is provided for this parameter, the first Transition (alphabetically by the value of the Trigger field) in the State for which the value of the Default property is "yes" is used.	No

sys_TouchParentFolderEffect

This Effect runs when a Content Item is added to or removed from a Folder. The Effect touches all other Content Items in the Folder (and in any Subfolders) that are in a Public or Quick Edit State, updating the Last Modified Date to the current date. Touching these Content Items ensures that they will be re-published during the next Incremental Publish run.

sys_UnpublishMandatory

This Effect is used in processing mandatory Relationships. It prevents Transition of the Content Item that activated the Effect from a Public State unless the other Content Item in the Relationship is already in a non-Public State.

If the value of the forceTransition parameter is "yes", and either a Transition for which the value of the Trigger field matches the value of the appropriate TransitionName parameter or a Default Transition is specified from the current State of the associated Content Item to a non-Public State of that Content Item's Workflow, the associated Content Item will be Transitioned to the Public State along with the original Content Item. If the associated Content Item cannot be Transitioned to a non-Public State, Rhythmyx generates an error and the original Content Item is not allowed to make the Transition.

Parameters

Name	Description	Required?
forceTransition	Boolean flag ("yes" or "no") that specifies whether the Content Item associated in the Relationship will be forced to make the Transition, if the Transition is possible. If the value of this parameter is "no" and the associated Content Item is already in a non-Public State, the Transition of the original Content Item fails.	Yes
ownerTransitionName	Internal name (value of the Trigger field of the Edit Transition page) of the Transition to use to Transition the associated Content Item to another State if that Content Item is the Owner in the Relationship. If no value is provided for this parameter, the first Transition (alphabetically by the value of the Trigger field) in the State for which the value of the Default property is "yes" is used.	
dependentTransitionName	Internal name (value of the Trigger field of the Edit Transition page) of the Transition to use to Transition the associated Content Item to another State if that Content Item is the Dependent in the Relationship. If no value is provided for this parameter, the first Transition (alphabetically by the value of the Trigger field) in the State for which the value of the Default property is "yes" is used.	No

sys_Validate

Use this Effect to perform validation on Rhythmyx objects in Relationships. Use conditional statements for the Effect to perform the validation.

You must define conditions inverse to the condition you want to validate. For example, if you want to implement a Content Item validation (the request fails if the owner is not a Content Item), you would you would make this Effect the first Effect in the Relationship and define the following conditional statement for the Effect:

```
PSXContentItemStatus/CONTENTSTATUS.OBJECTTYPE != 1
```

(The objecttypeid of Content Items is "1".)

Parameters

Name	Description	Required?
errorMessage	The text Rhythmyx displays if the object fails the validation. This text can be internationalized.	No

sys_ValidateFolder

NOTE: This Effect is deprecated in Rhythmyx Version 5.7. New installations of Rhythmyx 5.7 and later do not include a registration for this Effect. During upgrade from earlier versions to Rhythmyx Version 5.7, the Effect will be flagged as deprecated in the Extension registration.

This Effect validates that the Owner in the Relationship is a Folder and that the Dependent in the Relationship, if it is a Folder, has a name unique among all child Folders of the Owner. If the object fails this validation, Rhythmyx displays an error message.

Parameters

None

Index

T

U

W